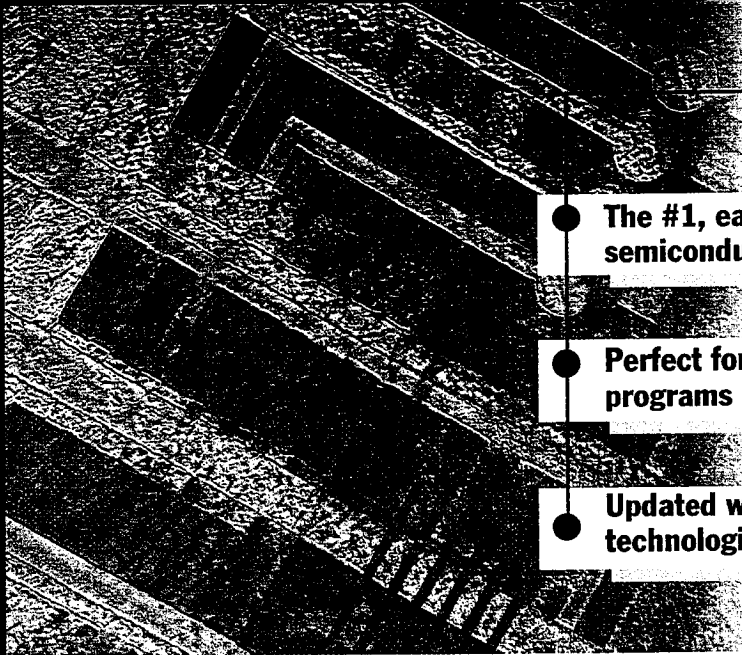


**The #1 book in the industry — now revised & updated!**



- **The #1, easy-to-read, math-free introduction to semiconductor processing**

- **Perfect for training, teaching, & vo-tech programs**

- **Updated with new cleaning techniques, packing technologies, & fabrication methods**

**F O U R T H   E D I T I O N**

# **Microchip Fabrication**

**A Practical Guide to  
SEMICONDUCTOR PROCESSING**

**PETER VAN ZANT**

DEF085330

**Library of Congress Cataloging-in-Publication Data**

Van Zant, Peter.

Microchip fabrication : a practical guide to semiconductor processing / Peter Van Zant.—4th ed.

p. cm.

Includes bibliographical references and index.

ISBN 0-07-135636-3

1. Semiconductors—Design and construction. I. Title.

TK7871.85.V36 2000

621.3815'2—dc21

00-02317

**McGraw-Hill**



A Division of The McGraw-Hill Companies

Copyright © 2000, 1997, 1984 by The McGraw-Hill Companies, Inc. All rights reserved. Printed in the United States of America. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher.

4 5 6 7 8 9 0 DOC/DOC 0 9 8 7 6 5 4 3 2

ISBN 0-07-135636-3

*The sponsoring editor for this book was Stephen Chapman and the production supervisor was Sherri Souffrance. It was set in Century Schoolbook by Pro-Image Corporation.*

*Printed and bound by R. R. Donnelley & Sons Company.*



This book is printed on recycled, acid-free paper containing a minimum of 50% recycled, de-inked fiber.

Information contained in this work has been obtained by The McGraw-Hill Companies, Inc. ("McGraw-Hill") from sources believed to be reliable. However, neither McGraw-Hill nor its authors guarantee the accuracy or completeness of any information published herein, and neither McGraw-Hill nor its authors shall be responsible for any errors, omissions, or damages arising out of use of this information. This work is published with the understanding that McGraw-Hill and its authors are supplying information but are not attempting to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought.

DEF085331

each process in turn requires a number of steps and substeps. A speculative process for a 64Gb CMOS device might require 180 major steps, 52 clean/strip, and up to 28 masks.<sup>1</sup> Yet all of the major steps are one of the four basic operations. Figure 4.11 lists the basic operations and the principle process options used for each. In this section, the building of a simple device, an MOS silicon gate transistor, is illustrated to explain a manufacturing sequence. The functions of the individual parts of this type of transistor and the operation of the transistor are explained in Chapter 14.

### Circuit design

Circuit design is the first step in creation of a microchip. A circuit designer starts with a block functional diagram of the circuit such as the logic diagram in Fig. 4.12. This diagram lays out the primary functions and operation required of the circuit. Next, the designer translates the functional diagram to a schematic diagram (Fig. 4.13). This diagram identifies the number and connection of the various circuit components. Each component is represented by a symbol. Accompanying the schematic diagram are the electrical parameters (voltage, current, resistance, etc.) required to make the circuit work.

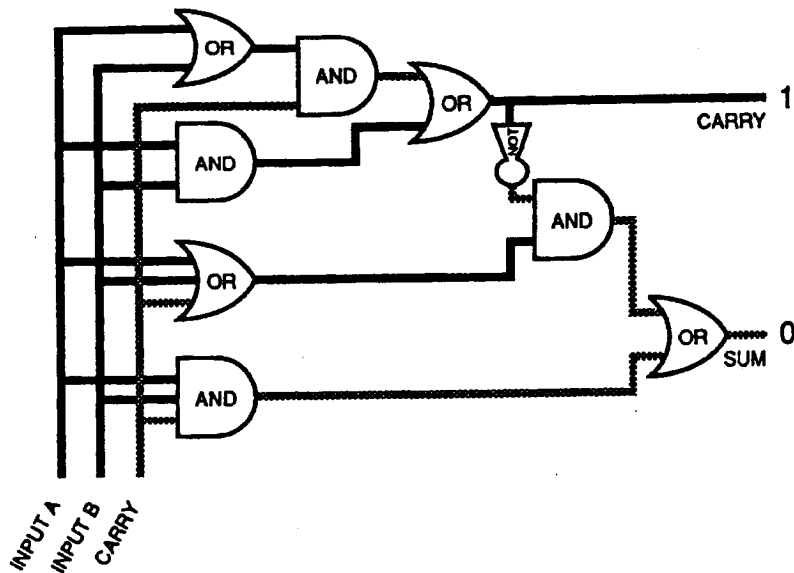
The third step, circuit layout, is unique to semiconductor circuits. Circuit operation is dependent on a number of factors, including material resistivity, material physics, and the physical dimensions of the individual component "parts". Also the placement of the parts relative to each other is another factor. All these considerations dictate the physical layout and dimensions of the part/device/circuit. Layout starts with using sophisticated computer-aided design (CAD) systems to translate each circuit component into the physical shape and size. Through the CAD system, the circuit is built, exactly duplicating the final design. The result is a composite picture of the circuit surface showing all of the sublayer patterns. This drawing is called a *composite* (Fig. 4.14). The composite drawing is analogous to the blueprint of a multistory office building as viewed from the top and showing all of the floors. However, the composite is many times the dimensions of the final circuit.

Both buildings and semiconductor circuits are built one layer at a time. Therefore it is necessary to separate the composite drawing into the layout for each individual layer in the circuit. Figure 4.14 illustrates the composite and individual layer patterns for a simple silicon gate MOS transistor.

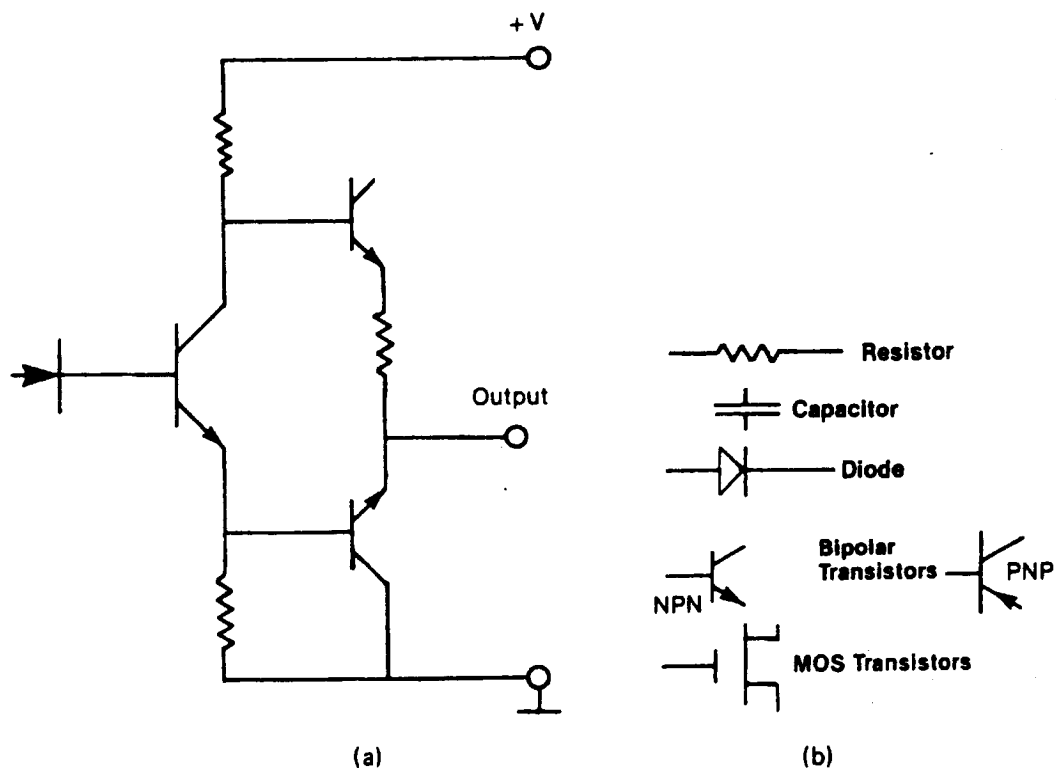
Each layer drawing is digitized (digitizing is the translation of the layer drawings to a digital data base) and plotted on a computerized X-Y plotting table.

Basic Operation	Process	Options
Layering	Oxidation	Atmospheric High Pressure Rapid Thermal Oxidation (RTO)
	Chemical Vapor Deposition (CVD)	Atmospheric Pressure Low Pressure (LPCVD) Plasma Enhanced (PECVD) Vapor Phase Epitaxy (VPE) Metalorganic CVD (MOCVD)
	Molecular Beam Epitaxy (MBE)	
	Physical Vapor Deposition (PVD)	Vacuum Evaporation Sputtering
Patterning	Resist	Positive Negative
	Exposure Systems	Contact Proximity Scanning Projection Stepper
	Exposure Sources	High Pressure Mercury X-Rays E-Beams
	Imaging Processes	Single Layer Resist Multilayer Resist Antireflecting layers Off-Axis Illumination Annular Ring Illumination Planarization Contrast Enhancement
	Etch	Wet Chemistry-Liquid/vapor Dry (Plasma) Lift-Off Ion Milling Reactive Ion Etch (RIE)
Doping	Diffusion	Open Tube-Horizontal/Vertical Closed Tube Rapid Thermal Processing (RTP)
	Ion Implantation	Medium/High Current Low/High Voltage (energy)
Heating	Thermal	Hot Plates Convection RTP
	Radiation	Infrared (IR)

Figure 4.11 Summary of wafer-fab operations/processes.



**Figure 4.12** Example functional logic design of a simple circuit.



**Figure 4.13** Example circuit schematic diagram with component symbols.

## Reticle and masks

The patterning process is used to create the required layer pattern and dimensions in and on the wafer surface. Getting the pattern from the digitized pattern to the wafer surface requires several steps. For the photo processes, there is an intermediate step called a reticle. A



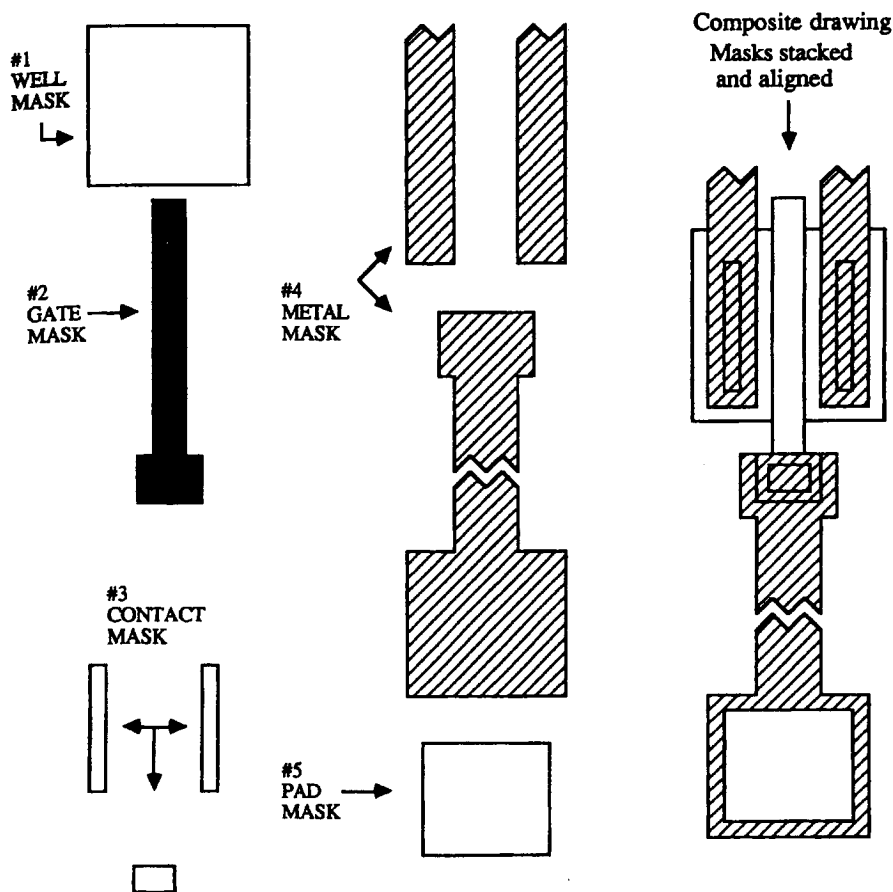


Figure 4.14 Composite and layer drawings for 5-mask silicon gate transistor.

reticle is a "hard copy" of the individual drawing recreated in a thin layer of chrome deposited on a glass or quartz plate (Fig. 4.15a). The reticle may be used directly in the patterning process or may be used to produce a photomask. A photomask is also a glass plate with a thin chrome layer on the surface. After production, it is covered with many copies of the circuit pattern (Fig. 4.15b). It is used to pattern a whole wafer surface in one pattern transfer. (Reticle and mask-making processes are detailed in Chapter 11.)

The process from circuit design to wafer patterning is shown in Fig. 4.15. Reticles and masks are produced in a separate department or are purchased from outside vendors. They supply the fabrication area with a separate set of reticles or set of masks (*mask set*) for each circuit type.

### Example Fabrication Process

The manufacture of a circuit starts with a polished wafer. The cross section sequence in Fig. 4.16 shows the basic operations required to

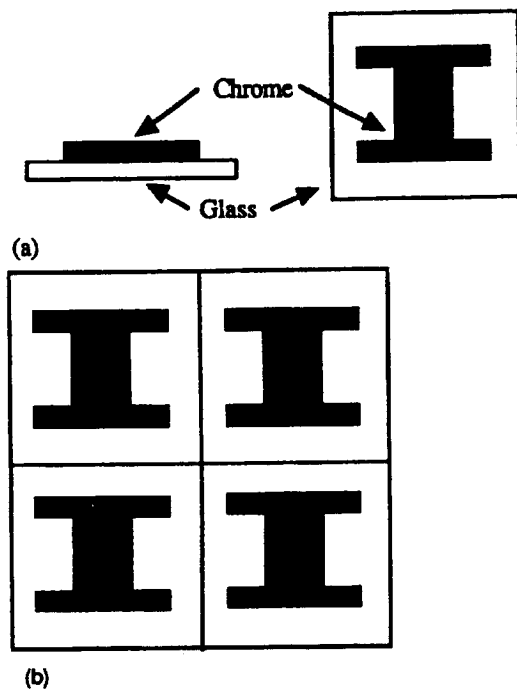


Figure 4.15 (a) Chrome on glass reticle. (b) Photomask of same pattern.

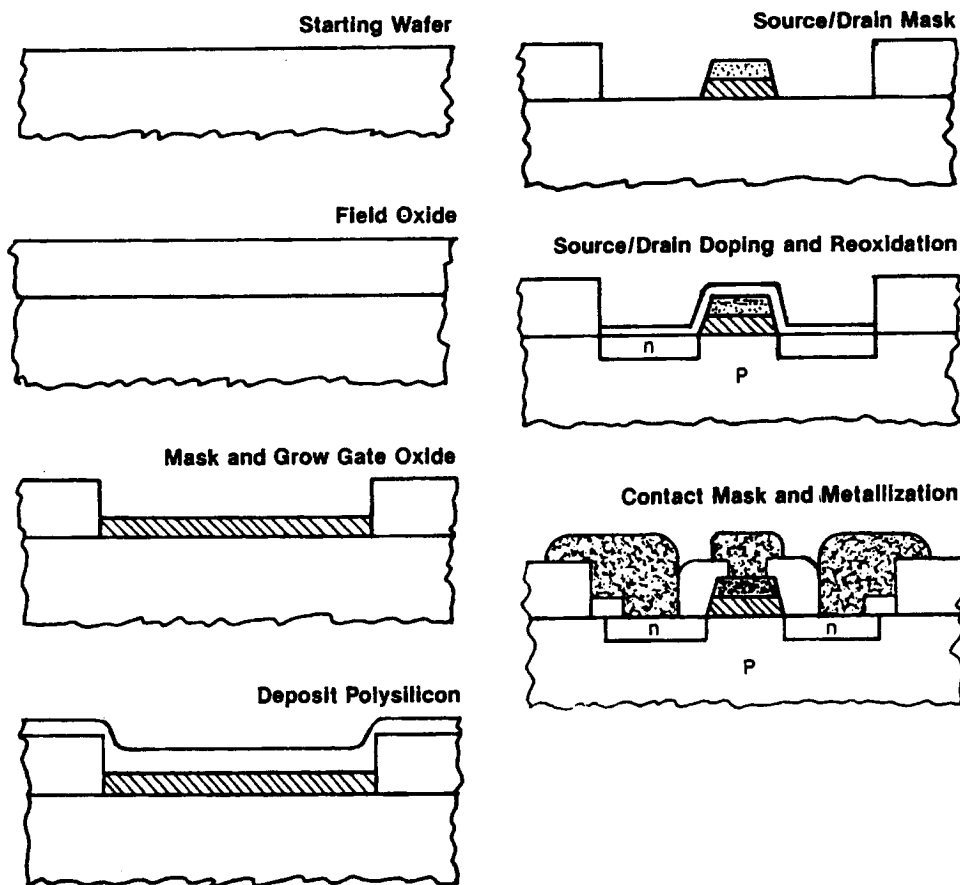


Figure 4.16 Silicon gate MOS process steps.

form a simple MOS silicon-gate transistor structure. Following is an explanation of each operation in the fabrication process.

**Step 1: Layering Operation.** The building starts with an oxidation of the wafer surface to form a thin protective layer and to serve as a doping barrier. This silicon dioxide layer is called the *field oxide*.

**Step 2: Patterning Operation.** The patterning process leaves a hole in the field oxide that defines the location of the source, gate, and drain areas of the transistor.

**Step 3: Layering Operation.** Next, the wafer goes to an silicon dioxide oxidation operation. A thin oxide is grown on the exposed silicon. It will service as the gate oxide.

**Step 4: Layering Operation.** In step 4, another layering operation is used to deposit a layer of polycrystalline (poly) silicon. This layer will also become part of the gate structure.

**Step 5: Patterning Operation.** Two openings are patterned in the oxide/polysilicon layer to define the source and drain areas of the transistor.

**Step 6: Doping Operation.** A doping operation is used to create a n-type pocket in the source and drain areas.

**Step 7: Layering Operation.** Another oxidation/layering process is used to grow a layer of silicon dioxide over the source/drain areas.

**Step 8: Patterning Operation.** Holes, called contact holes, are patterned in the source, gate, and drain areas.

**Step 9: Layering Operation.** A thin layer of conducting metal, usually an aluminum alloy, is deposited over the entire wafer.

**Step 10: Patterning Operation.** After deposition, the wafer goes back to the patterning area where portions of the metallization layer are removed from the chip area and the scribe lines. The remaining portions connect all the parts of the surface components to each other in the exact pattern required by the circuit design.

**Step 11: Heat Treatment Operation.** Following the metal patterning step, the wafer goes through a heating process in a nitrogen gas atmosphere. The purpose of the step is to "alloy" the metal to the exposed source and drain regions and the gate region to ensure good electrical contact.

**Step 12: Layering Operation.** The final layer of this device is a protective layer known variously as a *scratch* or *passivation layer* (not shown in Fig. 4.5). Its purpose is to protect the components on the chip surface during the testing and packaging processes, and during use.

**Step 13: Patterning Operation.** The last step in the sequence is a patterning process that removes portions of the scratch protection



## 82 Chapter Four

layer over the metallization terminal pads on the periphery of the chip. This step is known as the *pad mask* (not shown in Fig. 4.6).

The 12-step process illustrates how the four basic fabrication operations are used to build a particular transistor structure. The other components (diodes, resistors, and capacitors) required for the circuit are formed in other areas of the circuit as the transistors are being formed. For example, in this sequence resistor patterns are put on the wafer at the same time as the source/drain pattern for the transistor. The subsequent doping operation creates the source/drain *and* the resistors. Other transistor types, such as bipolar and silicon gate MOS, are formed by the same basic four operations, but using different materials and in different sequences.

### Chip Terminology

Figure 4.17 is a photomicrograph of an MOS medium-scale integration (MSI) integrated circuit. This level of integration was chosen so that the surface details could be seen. The components of higher-density circuits are so small that they cannot be distinguished on a photomicrograph of the entire chip. The chip features are:

1. A bipolar transistor
2. The circuit designation number
3. Bonding pads for connecting the chip into a package
4. A piece of contamination on a bonding pad
5. Metallization lines
6. Scribe (separation) line
7. Unconnected component
8. Mask alignment marks
9. Resistor

### Wafer Sort

Following the wafer-fabrication process comes a very important testing step, wafer sort. This test is the report card on the fabrication process. During the test, each chip is electrically tested for electrical performance and circuit functioning. Wafer sort is also known as *die sort* or *electrical sort*.

dry combinations as the mainstream photoresist removal process. Plasma stripping is used to remove hardened resist layers. A follow-on wet strip step is used to remove the residuals not removed by the plasma.

### Post-ion implant and plasma etch stripping

Two problem areas are photresist removal after ion implant and after a plasma strip. Ion implant causes extreme polymerization of the resist and crusting of the top. Generally, the resist is removed or reduced with a dry process, followed by a wet process. Post-plasma etch resist layers are similarly difficult to remove. In addition, the etch process can leave residues, such as  $\text{AlCl}_3$  and/or  $\text{AlBr}_3$ , that react with water or air forming compounds that corrode the metal system.<sup>34</sup> Low temperature plasma can remove the offending compounds before they take on a corrosive chemistry. Another approach is to add halogens to the plasma atmosphere to minimize the formation of the insoluble metal oxides. This is another instance of setting process parameters to achieve efficient processing (resist removal) without inducing wafer surface damage or metal corrosion.

### Final Inspection

The final step in the basic photomasking process is a visual inspection. It is essentially the same procedure as develop inspect, with the exception that the majority of the rejects are fatal (no rework is possible). The one exception is contaminated wafers that may be re-cleaned and reinspected. Final inspection certifies the quality of the outgoing wafers and serves as a check on the effectiveness of the develop inspection. Wafers that should have been identified and pulled from the batch at develop inspect are called *develop inspect escapes*.

The wafers receive a first surface inspection in incident white or ultraviolet light for stains and large particulate contamination. This inspection is followed by a microscopic or automatic inspection for defects and pattern distortions. Measurement of the critical dimensions for the particular mask level is also part of the final inspection. Of primary interest is the quality of the etched pattern with underetching and undercutting being two parameters of concern. The table in Fig. 9.29 is a list of typical causes of wafer rejection found in the final inspection.

### Mask Making

In Chapter 5, the steps of circuit design were detailed. In this section, the process used to construct a photomask or reticle is examined. Orig-

Possible Process Cause	Contamination	Misalign	Undercut	Incomplete Etch	Wrong Mask	Pin Holes	C.D.'s	Visual Reject
Contaminated Etch	x		x	x				
Contaminated Stripper	x							
Contaminated H <sub>2</sub> O	x							
Insufficient Rinse	x		x					
No Wet Agent				x				
Under Etch				x			x	
Over Etch			x				x	
Wrong Etch			x	x			x	
Hard Bake Too High			x	x			x	
Poor Develop				x			x	
P <sub>2</sub> O <sub>5</sub> & SiO <sub>2</sub>			x				x	
B <sub>2</sub> O <sub>3</sub> & SiO <sub>2</sub>				x			x	
Low Hard Bake			x					
Develop Inspect Escapes		x	x	x	x	x		

Figure 9.29 Final inspect rejects and process causes.

inally the masks were made from emulsion-coated glass plates. The emulsions are similar to those found on camera film. These masks were vulnerable to scratches, deteriorated during use, and were not capable of resolving images in the sub 3 micron range. Masks for most modern work use a chrome on glass technology. This mask-making technology is almost identical to the basic wafer-patterning operation (Fig. 9.30). In fact the goal is the same, the creation of a pattern in the thin chrome layer on the glass reticle surface. The preferred materials for mask/reticles are borosilicate glass or quartz, which have good dimensional stability and transmission properties for the wavelengths of the exposing sources. Chrome layers are in the 1000-Å range and deposited on the glass by sputtering (see Chapter 12). Advanced mask/reticles use layers of chromium, chromium oxide, and chromium nitride.<sup>35</sup>

Mask/reticle making follows a number of different paths depending on the starting exposure method (pattern generation, laser, e-beam) and the end result (reticle or mask) (Fig. 9.31). Flow A shows the process for making a reticle using a pattern generator, which is an older technology. A pattern generator consists of a light source and a series of motor-driven shutters. The chrome-covered mask/reticle, with a layer of photoresist is moved under the light source as the shutters are moved and opened to allow precisely shaped patterns of

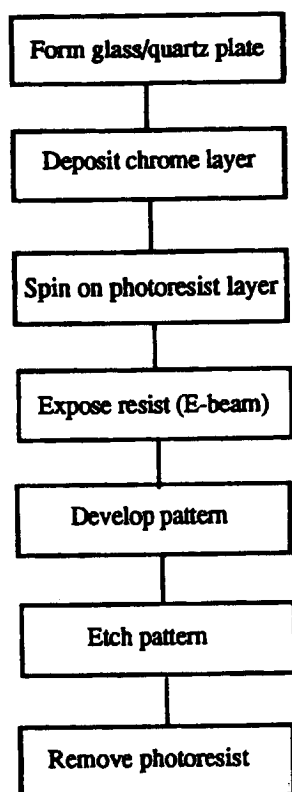


Figure 9.30 Major steps in mask/reticle plate processing.

light to shine onto the resist creating the desired pattern. The reticle pattern is transferred to the resist-covered mask blank by a step-and-repeat process to create a master plate. The master plate is used to create multiple working mask plates in a contact printer. This tool brings the master into contact with a resist-covered mask blank and has a UV light source for transferring the image. After each of the exposure steps (pattern generation, laser, e-beam, master plate expose, and contact print), the reticle/mask is processed through development, inspection, etch, strip, and inspection steps that transfer the pattern permanently into the chrome layer. Inspections are very critical since any undetected mistake or defect has the potential of creating thousands of scrap wafers. Reticles for this use are generally 5 to 20 times the final image size on the mask.<sup>36</sup>

Advanced products with very small geometries and tight alignment budgets require high quality reticle and/or masks. The reticles and masks for these processes are made with lasers or e-beam direct write exposure (Flow A & B). Laser exposure uses a wavelength of 364 nm making it an I-line system. It allows using standard optical resists and is faster than e-beam. Direct write laser sources are turned on and off with an acousto-optical modulator (AOM).<sup>37</sup> In all cases, the reticle or mask is processed to etch the pattern in the chrome.



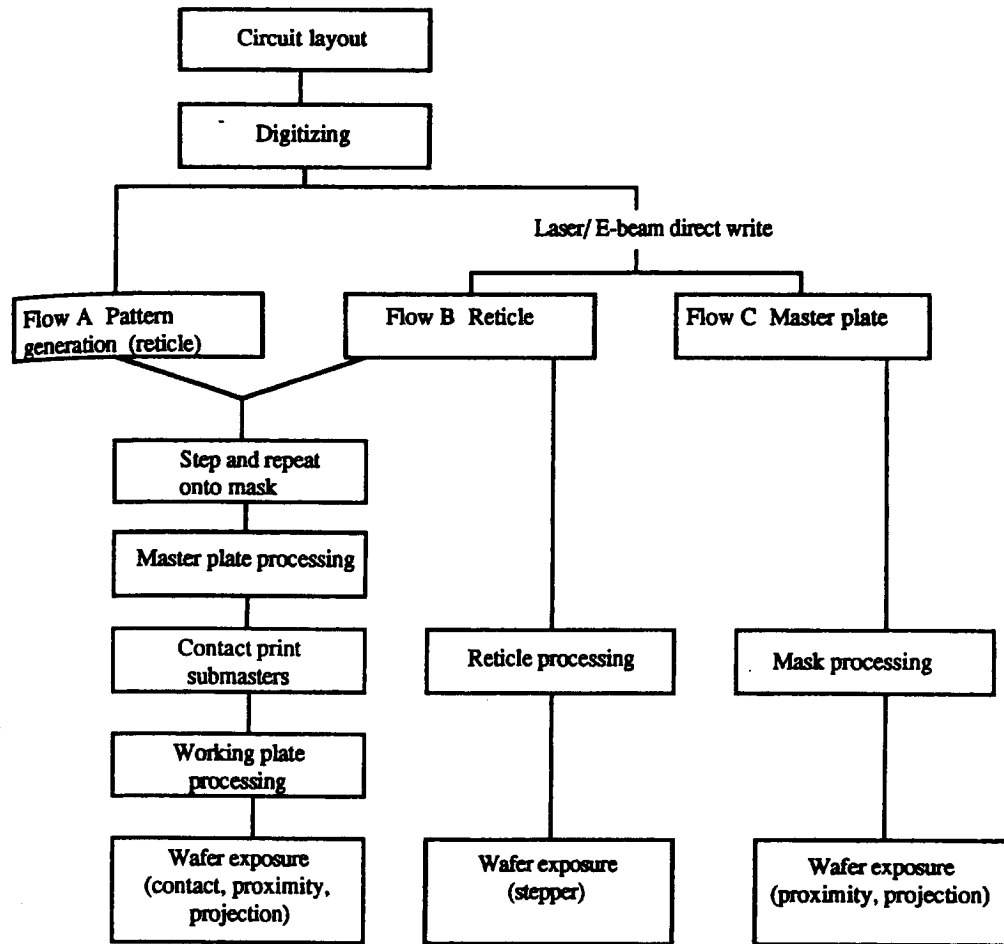


Figure 9.31 Mask/reticle-making processing flows.

Other mask/reticle process flows may be employed. The reticle in Flow A may be laser/e-beam generated or the master plate may be laser/e-beam generated.

VLSI and ULSI-level circuits require virtually defect-free and dimensionally perfect masks and reticles. Critical dimension (CD) budgets from all sources are 10% or better, leaving the reticles with a 4% error margin.<sup>38</sup> There are procedures to eliminate unwanted chrome spots and pattern protrusions with laser “zapping” techniques. Focused ion beams (FIB) is the preferred repair technology for small image masks and reticles. Clear or missing pattern parts are “patched” with a carbon deposit. Opaque or unwanted chrome areas are removed by sputtering from the beam.

### Summary

For VLSI and ULSI work, the resolution and registration requirements are very stringent. In 1977, the minimum feature size was 3



$\mu\text{m}$ . By the mid 1980s, it had passed the 1 micron barrier. By the 1990s 0.5 micron sizes were common with 0.35- $\mu\text{m}$  technology planned for production circuits. Circuit design projections call for minimum feature sizes less than 0.1  $\mu\text{m}$ .<sup>39</sup>

Chip manufacturers calculate several *budgets* for each circuit product. A *CD* (critical dimension) *budget* calculates the allowable variation in the image dimensions on the wafer surface. For products with submicron minimum feature sizes, the CD tolerances are 10 to 15%.<sup>40</sup> Also of concern is the critical defect size relative to the minimum feature size. These two parameters are brought together in an *error budget* calculated for the product. Error budgets for various DRAM products are illustrated in Fig. 9.31. An *overlay budget* is the allowable accumulated alignment error for the entire mask set. A rule of thumb is that circuits with micron or submicron feature sizes must meet registration tolerances of one-third the minimum feature. For a 0.35- $\mu\text{m}$  product, the allowable overlay budget is about 0.1  $\mu\text{m}$ .<sup>41</sup>

### Key Concepts and Terms

Alignment error budget	Negative resist developers
Develop inspect and rework	Plasma descum
Dry etch methods	Positive resist developers
Dry stripping	Puddle develop
Etch process	Resist development
Final inspect	Resist stripping
Hard bake methods	Spray develop
Hard bake process	Wet etch methods
Immersion develop	Wet strip chemicals
Mask making	

### Review Questions

1. Name the major methods of resist development.
2. What are the chemicals used to develop negative and positive resist?
3. What is the purpose of the hard bake step?
4. Name three methods used for hard bake.
5. What problems arise if the hard bake temperature is too low? Too high?



## CAD Tool Integration For ASIC Design: An End-Users Perspective

Pankaj Kukkal, Masato Imaizumi\*,  
and Hideaki Kobayashi  
Electrical and Computer Engineering  
University of South Carolina  
Columbia, South Carolina

### Abstract

The increasing complexity of VLSI design and the demand for quick-turnaround ASICs has forced the designer to choose the best CAD tools available from different vendors and integrate them into a customized and comprehensive CAD system. Vendors have developed comprehensive but open systems, called CAD frameworks, to ease the process of CAD tool integration.

This paper describes the ASIC development process and issues relating to CAD tool integration. In addition, state-of-the-art CAD frameworks and their impact on ASIC designers are described.

### I. Introduction

Application specific integrated circuits (ASIC) technology has brought a great revolution in VLSI system design. Before the advent of ASICs, systems were designed using a set of standard IC chips, whereas whole systems can now be designed into a single chip. ASICs are usually low volume products. Shorter design cycles are required to make the product cost competitive. Short design cycles imply increased dependence on CAD tools. To meet the demand of quick-turnaround ASICs, CAD tools have been developed to automate almost all the phases of ASIC design.

However, with increasing design complexities and design data, the flow of the design data through the CAD tools at various phases of the design cycle has become a cumbersome task for the ASIC designer. To relieve the designer of this burden, integrated CAD systems were developed. Initially, these systems were only a collection of tightly integrated tools. Adding and deleting tools was difficult, and usually these CAD systems were incompatible with each other. Such a closed environment is slow to evolve and the designer is restricted to a limited set of tools. Thus the concept of open design environments called CAD frameworks evolved. This concept is receiving wide attention in the area of computer aided design. A group of tool users, vendors, and system integrators, the CAD framework initiative (CFI), has set out to establish guidelines for a standard CAD framework. Thus we can look forward to a framework that encompasses all areas related to VLSI design such as specification capture, logic design, verification and testing.

### II. Application Specific ICs

#### Definition

ASICs are ICs designed for a specific application, in contrast to standard ICs that are used in different applications. ASICs are a class of ICs that are in between the domain of software programmable generic components, for example micro-processors, and hardware programmable components, for

\* M. Imaizumi is a Visiting Researcher to the Department of Electrical and Computer Engineering on leave from LSI Project, NKK Corporation, 2-6-3 Hitotsubashi, Chiyodaku, Tokyo 101, Japan.

example PLDs [KEUT,'89]. They can also be called custom-built ICs.

#### Characteristics

State-of-the-art electronics products such as digital signal processors, voice synthesizers, automatic focusing systems in cameras, etc. use ASICs. For prototypes, the circuits are designed by using a set of standard IC products but later for commercial products they are redesigned as an ASIC chip. System development cost is reduced by incorporating different components of the system, for example system controllers, RAMs, ROMs, PLDs, etc., into an ASIC. This technology also reduces the size of printed circuit boards thus reducing production cost. However, once systems are integrated into a chip, correcting even a small error, requires re-designing and re-fabricating the entire chip. Also, reduced node accessibility creates testing problems. Testing currently occupies 30-50 percent of the production cost of ASIC chips [LEUN,'88].

The ASIC designer uses a lot of CAD tools to reduce ASIC design time but has to sacrifice flexibility with regards to minimizing area and maximizing speed. A survey of ASICs designed for use by AT&T summarizes some common characteristics of ASICs [KEUT'89].

- *Control dominated* - Design of the control circuitry in the ASIC takes up a majority of circuit area and design time.
- *Arithmetic Structures* - Few large arithmetic circuits are present in an ASIC.
- *Speed/Area* - Even though high speed and component density are achievable, most ASICs use under 10,000 logic (non-memory) transistors and operate at no more than 10 MHz.
- *Regularity of structure* - ASIC designers use as many regular structures for eg. library cells, gate arrays, etc. as possible.
- *Analog interfaces* - Many asynchronous and analog interfaces are used.

### III. ASIC Design Styles and Methodology

#### Design Styles

Designers can employ various design styles to design ASICs. Some of the design styles and trade-offs involved are described below.

*Gate arrays* - In this style gates are pre-arranged, with space for channel routing. These gates are then interconnected by customizing metal layers. All levels of masks, except the metal interconnections, are predefined so that the wafer can largely be pre-fabricated. This prefabrication of wafers, called masterslices, is the main reason for fast turn-around time of prototype gate arrays. Due to its limited design freedom, the chip size of the gate array is typically two or three times that of a handcrafted design [LEUN,'88].

*Sea of gates* - This style is similar to gate arrays but has no

KBSC000031

predefined routing channels. So, sea-of-gates is also called "channel less gate array" and is more effective in area reduction than gate arrays.

**Standard cell** - Standard cells, for example, basic cells like inverters, NAND gates, Flip-Flops, etc., are pre-designed and are a part of a cell library. Placement and routing can both be customized. Usually all cells are of fixed height and a channel router is used to accomplish the interconnections. Pre-fabricated wafers are not used in this style.

**PLDs** - Programmable logic devices include PLAs, which are generated by automatic PLA generators and are usually used to implement system controllers.

**Custom cells** - Custom cells are designed to implement special functions that are not available in the library. If these custom cells are designed according to standard cell height specifications, then they can be added to the cell library and used with standard cells.

**Silicon compilers** - Once silicon compilers come of age the designer need only provide behavioral descriptions of the circuit and the corresponding hardware will be synthesized on silicon automatically. Presently, silicon compilers are used for generating megacells [LEUN, '88].

Usually, an ASIC is a mixture of modules designed using different design styles. For example, IBM has developed a design system that allows complete mixing of standard cell and gate array functions [LEUN, '88]. Thus we are moving towards an era where changing from one design style to another will be easy. This implies that designers will no longer be required to make tradeoffs between the various design styles. An integrated design environment will provide the designer with a complete solution to intermixing various styles.

#### Design methodology

Various aspects of VLSI design can be partitioned hierarchically into levels of abstraction. Design tools are used to implement the design at these levels. Table 1 describes an example of hierarchical levels, levels of abstractions, and CAD tools required to implement the design at these levels. A methodology is a sequence of design steps that links the design process from specification capture to mask layout. There are basically two types of hierarchical design methodologies: top-down (see Figure 1) and bottom-up. In top-down design methodology, a design is first described in general terms at some high level of design abstraction. Designers then recursively decompose and elaborate the design. In the bottom-up method the most detailed parts are designed first and then global layout is determined by combining these parts. Today, most designers employ the top-down design approach.

#### IV. ASIC Development Process

The ASIC development process is divided into four major areas. The design process being an important part of the overall ASIC development process, is presented here in greater detail.

**Specification Capture** - During specification capture, the objective or goal of designing the chip is decided. The performance characteristics and functionality of the chip are set forth. All constraints regarding the environment (i.e. all external factors affecting the chip in any possible way) are precisely defined [CAVI, '90].

**Design Synthesis and Verification** - This stage of ASIC development encompasses all the design phases from system design to the final mask layout. It can be viewed as a process of successive transformations from one hierarchical level to another.

Different phases of design synthesis and verification for a top-down design approach are listed as follows:

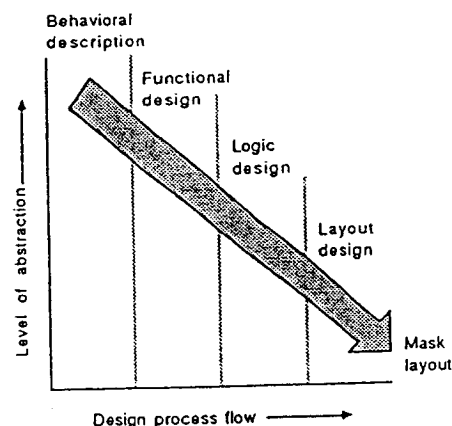


Figure 1. The top-down design process.

- System is designed according to the specifications.
- Decisions on fabrication process, for example the technology to be used, are made.
- Functional partitioning of the system into modules is done.
- Specifications for each module are set at some level of abstraction as shown in Table 1.
- Through a series of mapping and translation steps the design is taken down through the hierarchical levels, as shown in Table 1, to the logic design level.
- Logic is verified.
- The design is mapped down to the layout design level.
- Inverse mapping (mapping from a lower level to a higher level) is done and the layout is verified for consistency with the logic design.
- Logic and timing verification is performed at the layout level.
- Mask layout for the chip is obtained in a standard format, for example CIF, and sent to the fabrication house.

Table 1. Hierarchical and abstraction levels.

Hierarchy Levels	Abstraction Levels	CAD Tools
System	Timing behavior, pin assignments	Flowcharts, Block-diagrams, High level languages
Architecture	Organization of functional blocks	HDL's, Floorplanning, Block diagrams
Register transfer	Developing specifications for functional modules	Synthesis, Simulation, Verification, Test analysis
Logic	Boolean functions, Gate level circuits	Schematic entry, Simulation, Verification.
Transistor	Electrical properties of transistor circuits	SPICE, Timing verification
Layout	Geometric constraints	Layout editor, DRC and ERC programs, Netlist extractor, Placement and Routing tools

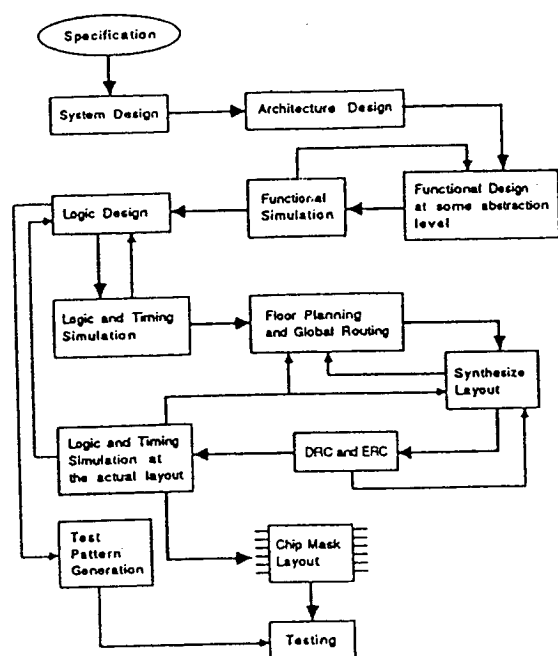


Figure 2. ASIC design flow.

Figure 2 depicts a typical ASIC design process. It can be seen from the figure that backtracking and iteration are required throughout the design process.

**Fabrication** - VLSI chips are fabricated by a complex series of about 100 or more steps. These steps create transistor parts, circuit elements, insulating layers, and metallized paths on silicon. Some major steps are thermal oxidation, lithography, etching, ion implantation, thermal redistribution, insulation, and metallization [STRO,'88].

**Testing and Verification** - Design verification techniques are used to qualify the ASIC to be marketed/supplied. At this stage the fabricated chip is tested for the specified external environment under accelerated stress conditions. Other tests to detect manufacturing defects are also carried out. Undetected errors in an ASIC can be very costly due to additional manufacturing costs and delay to market.

#### V. CAD Tools for ASIC Design

It is apparent from Table 1 and Figure 2 that a number of CAD tools are required to accomplish ASIC design tasks. These tools are classified as follows. (Examples for each class are taken from the UC Berkeley VLSI tool suite [SCOT,'85].)

**Mapping Tools** - These are used for schematic capture, interactive layout, graphical entry, text entry, placement and routing. For example, MAGIC layout editor is used for interactive layout, PEG is a text entry tool used for obtaining Boolean equations corresponding to a text that describes state diagrams.

**Translation Tools** - These are used for transforming files from one format to another to ensure compatibility between tools. For example, EXT2SIM is a tool for transforming the output of MAGIC layout editor to a format that is readable by simulator tools like ESIM and CRYSTAL.

**Validation Tools** - These tools check for violations of design rules. For example, the built-in Design Rule Checker (DRC) in the MAGIC layout editor concurrently checks for design rule

(design rules depend on technology and fabrication process) violations while the designer is laying out the design. There are CAD tools that perform Electrical Rule Checking (ERC) and check for connectivity of all the elements of the design.

**Verification Tools** - Logic and timing simulation on designs are performed by these tools. For example, ESIM is used for logic verification, CRYSTAL and SPICE are used for Timing verification.

**Optimization Tools** - Optimization tools are used to optimize area and speed. Area can be optimized by either layout compaction or reduction in the logic required to implement the design. For example, PLEASURE is used for PLA folding, and ESPRESSO is used for minimizing Boolean equations. Speed optimization depends largely on the designers expertise.

Along with the CAD tools the ASIC designer requires cell libraries. Cell libraries are a collection of primitive components, which can be individual transistors, logic gates or entire subsystems. These libraries may be traditional standard cells, gate array building blocks, parameterized cells such as those synthesized by silicon compilers, or sophisticated cells generated by module generators [NEWT,'86].

#### Invoking and Executing CAD tools

Some basic terms required for this discussion are defined [FIDU,'90] as:

- **Process** is a specific combination of tools and/or other processes that perform a design function.
- **Task** is an abstraction of a design function, e.g., simulation. Tasks are performed by invoking specific processes.

Whatever design methodology/style is employed, an ASIC designer needs to perform a sequence of tasks by using a set of CAD tools and cell libraries. A designer needs to invoke and execute each CAD tool required for ASIC design. This implies that he needs to select an appropriate tool for the given task and then execute that tool so that it accomplishes exactly what needs to be done. To select a tool, the designer needs to know the various available tools for the given task and then choose the best. This can only be done if the designer knows the exact functionality of all the CAD tools. The designer also needs information on the versions of CAD tools available. Thus many decisions are required even before invoking a tool. To execute the selected tool the designer needs to know the exact syntax for tool invocation, and semantics of the tool.

The designer also has to manage all design data files related to the tools. As most tools are incompatible with regards to input and/or output file formats the designer needs to translate these files [BUSH,'89]. For example, to run a simulation on a layout, geometrical information needs to be extracted from the layout and then converted to a file format that can be input to a simulator. The whole design, from behavioral description to mask layout, needs to be steered by the designer using the CAD tools. The designer can be relieved of these painstaking tasks if the following information about all the tools is embedded in the system:

- **Data requirements**; type, access modes, format, etc.
- **Argument definitions**; format, required/optional, purpose, etc.
- **Tool commands**; syntax, arguments, purpose, effect, etc.
- **Resource requirements**; regarding CPU time, memory, etc.
- **Description of tool functions**.

In [DANI,'89] an object-oriented approach to build models of CAD tools has been proposed. By binding a CAD tool to its representative model, they create a CAD tool knowledge object

KBSC000033



(CTKO). The CTKO represents the abilities of the tool to the designer, manages the low level programming details associated with the original tool, and provides a control mechanism between the tool and the designer.

#### CAD tool integration

CAD tools in the past decade have automated almost all the phases of the design process. The need of this decade is to integrate CAD tools into a design automation system. ASIC designers require an environment for integrating heterogeneous CAD tools while providing an open and distributed control mechanism. The need for CAD tool integration can be discussed with respect to four sub-topics as follows:

**Design data management** - With increasing design complexity, managing the input and output design files from the CAD tools is becoming extremely difficult. This is true especially when the data generated by CAD tools is enormous.

**Number of CAD tools** - CAD tool vendors are continuously supplying the designer with better CAD tools. Since many tools are incompatible in many ways, it takes a lot of time and effort to integrate them into a comprehensive design environment. This process takes a lot of time and effort especially when the CAD tools have different user-interfaces and file formats.

**Complexity of CAD tools** - Learning new CAD tools is not a trivial task; it involves a lot of time and effort. And if the designer does not continuously update his information about new and better tools he is left with a small set of CAD tools. To remain competitive the designer has to learn, master, and integrate new CAD tools to the existing CAD environment.

**Tool invocation and execution** - An integrated CAD environment will allow the designer to automatically invoke and execute CAD tools to perform a given task.

Three vital aspects of tool integration are visual, data, and control [SUNT,'90].

**Visual integration** - Visual integration implies that all the tools integrated should have the same look and feel. The interface between the designer and the tools should be homogeneous. This user interface must isolate the designer from the specifics of tool invocation and execution. The user must be able to choose from a list (usually icons) of tools or the execution environment must be able to choose one automatically based on the functionality of the tool. The interface should track and display the state of any on-going task including status of all related processes.

**Data integration** - Data integration can be classified as data linkage, data interchange, and data sharing as shown in Figure 3.

- **Data linkage** means establishing semantic relationships between pieces of information maintained by different tools. For example, Sun Microsystems's NSE Link Service 1.0 provides limited capability of this type [SUNT,'90].
- **Data interchange** means transferring information between tools in some mutually agreed upon representation. Examples of data interchange mechanism include window cut and paste, data import/export. Case Data Interchange Format (CDIF), is an example of a standard representation [SUNT,'90].
- **Data sharing** means that tools directly access data stored in a mutually accessible place. The Mentor Graphics' data repository approach is an example of data sharing that involves a common database schema and storage of data under a common data management system [WINK,'90].

**Control integration** - Control integration can be classified as interprocess control and meta control as shown in Figure 4.

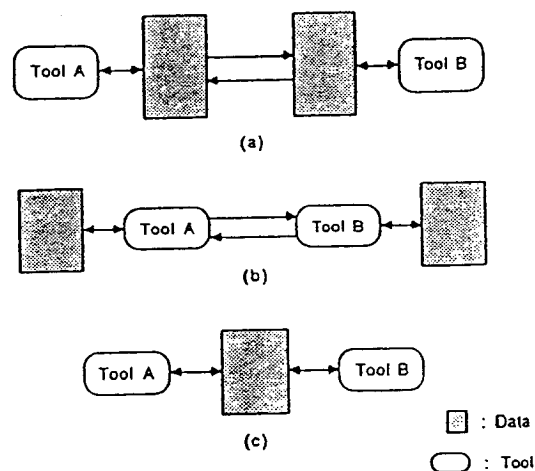


Figure 3. Data integration. (a) Data linkage. (b) Data interchange. (c) Data sharing.

- **Interprocess control** means that a tool can cause another tool to perform some action.
- **Meta control** means causing a sequence of tool invocations and requisite data interchanges to accomplish some specific task automatically. Meta control can be implemented in terms of interprocess control, in which an 'agent' tool co-ordinates actions performed by other tools.

#### VI. The CAD Framework

We have already seen the need for CAD tool integration. This need has forced tool developers to create suites of tools that shield the designer from as much lower level detail as possible. Traditionally these suites were tightly integrated into a design environment [SIEW,'83] [BROW,'83]. CAD tools were bound into a monolithic entity. A problem with this approach was that tools were difficult to add or delete. This led to the creation of CAD Frameworks that are intended to be open architectures that support a large population of heterogeneous tools. CAD frameworks may be used to configure a set of VLSI tools and to develop appropriate interfaces to support schematic capture, simulation, timing verification, and test generation for ASIC design [HARR,'90]. It should act like a conduit between the

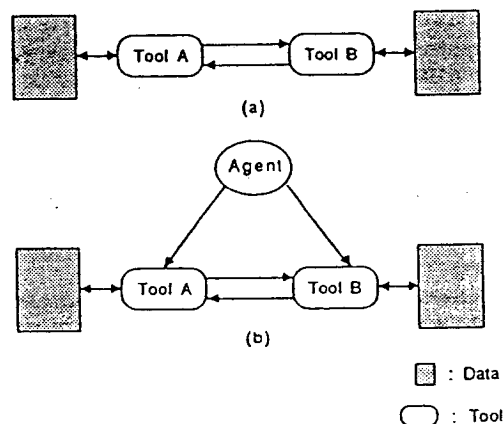


Figure 4. Control integration. (a) Inter-process control. (b) Meta control.



designer and the tools, matching the abilities of the tools to the needs of the designer. A framework should allow various tools to co-operate and work interactively with each other and with the designer. The designer need not learn all the subtle details of any tool; instead, the tools should present their general abilities to the designer as accurately and concisely as possible.

A generic CAD framework with its major components is shown in Figure 5. At a very high level the CAD framework can be viewed as having five major components described below [CAVI,'90]:

- A common user-interface which provides a consistent, graphical, and natural front end to the CAD tools.
- A design database containing design and library information which could be a centralized or distributed database.
- A design management database containing information on revisions, relationships, access authorizations, methodologies, and tools.
- A design data and process manager which utilizes the design management database to control design information and design process.
- CAD tools for all the phases of VLSI design.

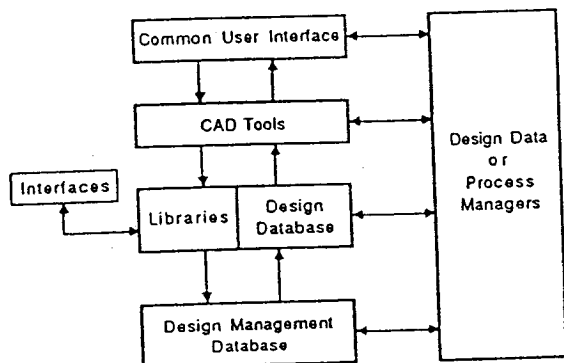


Figure 5. A generic CAD framework.

### Characteristics

Some important characteristics of CAD frameworks that directly effect the designer are summarized in this section.

A framework supports multiple users, thus the design can be divided among various designers. A distributed heterogeneous computing environment can be created thus allowing the designers to use tools that work under different computing environments.

The designer does not have to worry about the syntax of tool invocations and the semantics of tool commands since CAD frameworks will allow for automatic execution of tools. The designer is relieved of the burden of making available the proper data at the proper time to all the tools required in the design process. Once fully integrated into a framework all the tools will have the same look and feel making it easier for the designer to get accustomed to many different tools.

The designer can intermix design entry levels, for example hardware description languages (HDLs) and schematics, for specifying a design. CAD frameworks will allow for simulations of such intermixed design entries.

A framework supports different design styles and methodologies. It also allows an expert designer to record and save successful design methodologies, allowing less

experienced designers to create good designs using the knowledge of an expert designer. CAD frameworks combine top-down design with bottom-up schematic entry oriented design approach; this lets the designer tackle extremely complex IC designs. The designer can make arbitrary changes to the design at any stage and the framework will take care of design consistency by automatically propagating the effects of the changes to all related files.

Design complexity makes it impossible for the designer to foresee all situations where incompatibilities occur between the physical structure and the functional specifications. CAD frameworks will keep track of various versions of complex designs through all the phases of the development cycle while ensuring data integrity. Therefore, the designer can easily backtrack through all his previous design steps. He can also track the evolution of the design.

The framework eliminates long error-prone translations of data files between CAD tools since all the databases of the individual tools are either linked together by a common database or there is just a single database for all the tools.

CAD frameworks can be used for concurrent hardware and software design. Thus it will be possible to test software code on simulated hardware, permitting changes in either or both, to better meet the needs of the system [ADAM,'90]. It decreases the total turn-around time of the design process.

Until now, most of the decisions at every phase of the design were a matter of experience. With increasing complexities of the design process, just one wrong decision can prove fatal. The designer needs some decision support at the earlier stages of the design process. For example, if the designer decides on a particular floorplan he can get information about the affect of that floorplan on the final layout area of chip. Such decision support is provided by the framework.

CAD frameworks will allow the designers to work at a higher level of abstraction by allowing for simulations at the behavioral level. Designers need not become expert users of a baffling array of complex CAD tools, but instead need only be concerned with higher level design tasks. Designers can rectify potential problems before they happen at lower levels of design.

Knowledge required for good designs, such as the fundamentals of material properties, fabrication processes, device characteristics, microelectronics and circuit techniques, can also be built into the frameworks.

### CAD Framework Initiative

Some vendors have come up with commercially available frameworks, for example Mentor Graphics' Falcon Framework, DEC's PowerFrame and Cadence's Design Framework II. Though there is general agreement over the basic definition of a framework, there are many different ways to implement the idea.

The CAD Framework Initiative (CFI), a body of CAD tool users, vendors, and system integrators was formed to supervise these ongoing framework efforts and lay down industry acceptable guidelines for design frameworks [CADF,'90]. CFI's purpose is to provide industry standard interfaces to CAD framework components, thus reducing the cost of combining multi-sourced CAD tools into an effective CAD system.

The CFI group defines a framework [CADF,'90] as:

"A software infra-structure that provides an environment where CAD tools are developed, integrated, and operated."

It further specifies that through a CAD framework, a user should be able to launch and manage tools; create, organize, and manage data; graphically view the entire design process; and perform design management tasks such as configuration management, version management, etc..

KBSC000035

### CFI and CAD tool integration

CFI has come up with a seven layer conceptual model [FRAM,'90] of a framework (Figure 6). Out of these seven layers two layers (Levels 4 and 5) refer specifically to CAD tools. Level 4 allows for tool management which means that the tools have to be managed within the context of the overall design process. It specifies that the frameworks should supply models and services that describe inputs, outputs, options, invocation sequences, control files, etc. to manage tools. Usually tools can be integrated into any CAD system but the integration software has to deal with data translation between the tools. Level 5 allows for easy integration by separating the data descriptions from the tool integration programs.

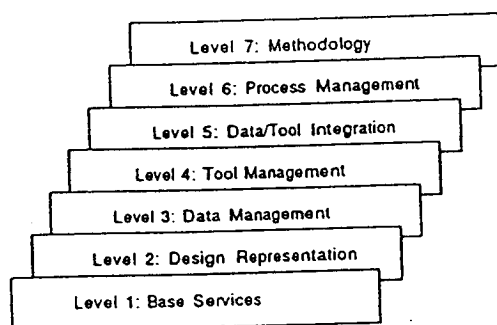


Figure 6. CFI's conceptual model of frameworks.

### VII. Conclusion

In this paper we have identified the ASIC development process and CAD tools required for an ASIC design process. Then we have shown the evolution of computer aided VLSI design from the point where individual CAD tools were available to perform some of the design tasks, to a unified design environment called the CAD Framework. CAD frameworks not only integrate CAD tools into an open environment but also automate the design process. We have identified the impact of CAD frameworks on the designer and thus hope that this paper helps the ASIC designers to introduce the benefits of CAD frameworks into their existing design environment.

We look forward to CAD frameworks that will support a fully automated design process. The designer need only be involved in the process of specification capture and feeding behavioral/functional specifications to the CAD system. By the end of this decade we hope that there will be accepted standards for integrating not only CAD tools but tools that will encompass all other IC related areas such as manufacturing, control systems, process and device modelling, etc.

### References

- [ADAM,'90] Adams, Charlotte, "CAD Frameworks: Putting it all together", *Military and Aerospace Electronics*, pp. 33-38, July 1990.
- [BROW,'83] Brown, H., et. al., "Palladio: An Exploratory Environment for Circuit Design", *IEEE Computer*, December 1983.
- [BUSH,'89] Bushnell, M.L. and Director, S.W., "Automated Design Tool Execution in the Ulysses Design Environment", *IEEE Trans. on Computer Aided Design*, vol. 8, no. 3, pp. 279-287, March 1989.
- [CADE,'90] TCC-Approved Draft Proposal, "CAD Framework Users, Goals and Objectives", 1990.
- [CAVI,'90] Cavin, R.K. and Hilbert, J.L. "Design of Integrated Circuits: Directions and Challenges", *Proceedings of the IEEE*, vol. 78, no. 2, pp. 418-435, Feb. 1990.
- [DANI,'89] Daniell, J. and Director, S.W., "An Object Oriented approach to CAD Tool Control within a Design Framework", *Proceedings IEEE 26th. Design Automation Conference*, pp. 197- 202, 1989.
- [FIDU,'90] Fiduk, K.W., et al., "Design Methodology Management -A CAD framework Initiative Perspective", *Proceedings IEEE 27th. Design Automation Conference*, pp. 278-283, 1990.
- [FRAM,'90] Cadence's catalogue, "Framework Technology for the 1990s", 1990.
- [HARR,'90] Harrison, D.S., et al., "Electronic CAD Frameworks", *Proceedings of the IEEE*, vol. 78, no. 2, pp. 393-417, Feb. 1990.
- [KEUT,'89] Keuterzer, Kurt, "Three Competing Design Methodologies for ASICs: Architectural Synthesis, Logic Synthesis and Module generation", *Proceedings IEEE 26th. Design Automation Conference*, pp. 308-313, 1989.
- [LEUN,'88] Leung, S.S., Fisher, P.D., and Shanblatt, M.A., "A conceptual framework for ASIC design", *Proceedings of the IEEE*, vol. 76, no. 7, pp. 741-755, July 1988.
- [NEWT,'86] Newton, A. R., and Sangiovanni-Vincentelli, A. L., "Computer-Aided Design for VLSI circuits", *IEEE Computer*, pp. 38-60, April 1986.
- [SCOT,'85] Scott, W. S., et. al., "1986 VLSI Tools: Still More Works by the Original Artists", *Report No. UCB/CSD 86/272*, Univ. of California, Berkeley, December 1985.
- [SIEW,'83] Siewiorek, D. P., et. al., "DEMETER-A Design Methodology and Environment", *Tech. report CMUCAD-83-5*, Electrical and Comp. Eng. Dept., Carnegie Mellon University, 1983.
- [STRO,'88] Strojwas, A. J., and Director, S. W., "VLSI: linking design and manufacturing", *IEEE Spectrum*, pp. 24-28, Oct. 1988.
- [SUNT,'90] "CASE Integration Frameworks", *SunTech Journal*, Nov. 1990.
- [WINK,'90] Winkler, E., "EDA Firms Find Several Roads to Frameworks", *Electronic News*, June 25, 1990.



# Dictionary of Computing

▼ The most comprehensive computing dictionary ever published

▼ More than 18,000 entries

DEF083932

**Limitation of Liability**

While the Editor and Publisher of this book have made reasonable efforts to ensure the accuracy and timeliness of the information contained herein, neither the Editor nor the Publisher shall have any liability with respect to loss or damage caused or alleged to be caused by reliance on any information contained herein.

Copyright © 1994 by International Business Machines Corporation. All rights reserved. Printed in the United States of America. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher.

7 8 9 0 DOC/DOC 9 9 8

ISBN 0-07-031488-8 (HC)  
ISBN 0-07-031489-6 (PBK)

*The sponsoring editor for this book was Daniel A. Gonneau and the production supervisor was Thomas G. Kowalczyk.*

*Printed and bound by R. R. Donnelley & Sons Company.*

**Tenth Edition (August 1993)**

This is a major revision of the *IBM Dictionary of Computing*, SC20-1699-8, which is made obsolete by this edition. Changes are made periodically to the information provided herein.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Comments may be addressed to IBM Corporation, Department E37/656, P. O. Box 12195, Research Triangle Park, NC 27709.

**International Edition**

Copyright © 1994 by International Business Machines Corporation. Exclusive rights by McGraw-Hill, Inc. for manufacture and export. This book cannot be re-exported from the country to which it is consigned by McGraw-Hill. The International Edition is not available in North America.

When ordering this title, use ISBN 0-07-113383-6.

**compound string**

[129]

**computer-aided design (CAD)**

**compound string** A type of string designed to simplify national language support by allowing text to be displayed without hard-coding language-dependent attributes such as character sets and text.

**compress** (1) In a character string, to reduce the space taken on a data medium by repetitive characters. (T) (2) To save storage space by eliminating gaps, empty fields, redundancy, or unnecessary data to shorten the length of records or files. (3) To move files, libraries, or folders together on disk to create a continuous area of unused space.

**compressed audio** Audio resulting from the process of digitally encoding and decoding up to 40 seconds of voice-quality audio for each individual videodisc, resulting in a potential for over 150 hours of audio per 12-inch videodisc. Synonymous with still-frame audio.

**compressed disk file** In System/36, a file that contains unprocessed records.

**compressed encoding** (1) A process in which a contiguous string of bits, characters, or data units is reduced to a shorter string in such a way that a second contiguous string, yielding the same short string, cannot be found and the process cannot be reversed. (2) A process in which variable-length messages are reduced to a shorter, fixed-length message.

**compressed pattern storage** Storage that holds the extended fonts for the 3800 printer.

**compressed video** Video resulting from the process of digitally encoding and decoding a video image or segment using a variety of computer techniques to reduce the amount of data required to represent the content accurately.

**compression** (1) The process of eliminating gaps, empty fields, redundancies, and unnecessary data to shorten the length of records or blocks. (2) In SNA, the replacement of a string of up to 64 repeated characters by an encoded control byte to reduce the length of the data stream sent to the LU-LU session partner. The encoded control byte is followed by the character that was repeated (unless that character is the prime compression character). See also compaction, string control byte. (3) In Data Facility Hierarchical Storage Manager, the process of moving data instead of allocated space during migration and recall in order to release unused space. (4) Contrast with decompression.

**compressor** An electronic device that compresses the volume range of a signal. See also compandor, expander.

**COM printer** A page printer that produces on a photographic film a microimage of each page. Synonymous with computer output microfilm printer. (T) (A)

**compromise** In computer security, a violation of the security policy of a system in which unauthorized intentional or unintentional disclosure, modification, or destruction, or loss, of an object, may have occurred.

**compromise net** A network, used in conjunction with a hybrid coil to balance a subscriber's loop, that is adjusted for an average loop length or an average subscriber's set, or both, to secure compromise (not precision) isolation between the two directional paths of the hybrid.

**compromising emanations** In computer security, unintentional intelligence-bearing signals that may convey data and that, if intercepted and analyzed, may compromise sensitive information being processed or transmitted by a computer system.

**COMPUSEC** Computer security.

**computational stability** The degree to which a computational process remains valid when subjected to effects such as errors, mistakes, or malfunctions. (A)

**compute mode** That operating mode of an analog computer during which the solution is in progress. Synonymous with operate mode. (T)

**computer** A functional unit that can perform substantial computations, including numerous arithmetic operations and logic operations without human intervention during a run. In information processing, the term computer usually describes a digital computer. A computer may consist of a stand-alone unit or may consist of several interconnected units. (T)

**computer abuse** In computer security, a willful or negligent unauthorized activity that affects the availability, confidentiality, or integrity of computer resources. Computer abuse includes fraud, embezzlement, theft, malicious damage, unauthorized use, denial of service, and misappropriation. See also information system abuse.

**computer-aided (CA)** Pertaining to a technique or process in which part of the work is done with the assistance of a data processing system. Synonymous with computer-assisted. (T)

**computer-aided design (CAD)** The use of a computer to design or change a product, tool, or machine, such as using a computer for drafting or illustrating.

**Note:** Sometimes, CAD and CAM are used together and expressed as CAD/CAM. (T)

DEF084073



**computer-aided engineering (CAE)**

[130]

**computer generation**

**computer-aided engineering (CAE)** Analysis of a design to check for basic errors, or to optimize manufacturability, performance or economy; for example, by comparing various possible materials or designs.

**Note:** Information from the CAD/CAM design database is used to analyze the functional characteristics of a part, product under design, and to simulate its performance under various conditions. (T)

**computer-aided industry (CAI)** The use of computer systems to assist in the operation of an industry. (T)

**computer-aided instruction (CAI)** The use of a computer to assist human instruction. (T) Synonymous with computer-assisted instruction.

**computer-aided manufacturing (CAM)** The use of computer technology to direct and control the manufacturing process. (T)

**computer-aided planning (CAP)** All activities for preparation of the basic data about production processes by usage of computer technology. (T)

**computer-aided publishing** Synonym for electronic publishing. (T)

**computer-aided quality assurance (CAQ)** The use of computer technology to plan, monitor and control processes, parts and products throughout all phases of the product life cycle; this includes an overall quality report system from design to field performance and from shop floor to the management. (T)

**computer-aided retrieval (CAR)** Systems that combine the document storage capabilities of micrographics with the indexing and retrieval capabilities of a computer database.

**computer-aided software engineering (CASE)** (1) The automation of well-defined methodologies that are used in the development and maintenance of products. These methodologies apply to nearly every process or activity of a product development cycle, examples of which include project planning and tracking, product designing, coding, and testing. (2) A set of computer-based development tools to automate certain portions of methodologies. Thus, CASE tools work within a methodology rather than compose a methodology themselves. See also CCASE, ICASE.

**computer-animated graphics** In multimedia applications, graphics animated by means of a computer, rather than videotape or film.

**computer architecture** (1) The logical structure and functional characteristics of a computer, including the

interrelationships among its hardware and software components. (T) (2) The organizational structure of a computer system, including hardware and software. (A) (3) See also hypercube, parallel processor architecture.

**computer-assisted** Synonym for computer-aided (CA). (T)

**computer-assisted instruction (CAI)** Synonym for computer-aided instruction.

**computer-assisted publishing** Synonym for electronic publishing. (T)

**computer-based training** Synonym for computer-assisted instruction.

**computer center** A facility that includes people, hardware, and software, organized to provide information processing services. Synonymous with data processing center, installation. (T) (A)

**computer conferencing** Computerized communication that allows people distant from each another to enter and receive text and graphic messages via interconnected terminals. (T) See also conference call, teleconferencing, videoconferencing.

**computer crime** (1) In computer security, a crime committed through the use of software or data. (T) (2) A crime committed through the use of software or data residing in a computer. (T) (A)

**computer cryptography** In computer security, the use of a cryptographic algorithm in a computer to perform encryption and decryption to protect information or to authenticate users, sources, or information.

**computer edit system** A video editing system, controlled by a computer and connected to machines for recording and playback.

**computer-dependent language** Synonym for computer-oriented language.

**computer fraud** (1) In computer security, a computer crime that involves deliberate misrepresentation or alteration of data in order to obtain something of value, usually for monetary gain. (T) (2) Deception by means of a computer, deliberately practiced in order to secure unfair or unlawful gain. (A)

**computer generation** A category in a historical classification of computers based mainly on the technology used in their manufacture; for example, first generation based on relays or vacuum tubes, the second on transistors, the third on integrated circuits. (T)

DEF084074

**operating time****[479]****operations analysis**

**operating time** (1) That part of operable time during which a functional unit is operated. (A) (2) Contrast with idle time.

**operating voltage indicator** On a calculator, a device giving a visual signal to indicate that the correct voltage is set for a main-powered machine or that the battery is insufficiently charged in a battery-powered machine. (T)

**operation** (1) A well-defined action that, when applied to any permissible combination of known entities, produces a new entity; for example, the process of addition in arithmetic; in adding five and three and obtaining eight, the numbers five and three are the operands, the number eight is the result, and the plus sign is the operator indicating that the operation performed is addition. (I) (A) (2) A defined action, namely, the act of obtaining a result from one or more operands in accordance with a rule that completely specifies the result for any permissible combination of operands. (A) (3) A program step undertaken or executed by a computer; for example, addition, multiplication, extraction, comparison, shift, transfer. The operation is usually specified by the operator part of an instruction. (A) (4) The event or specific action performed by a logic element. (A) (5) An action performed on one or more data items, such as adding, multiplying, comparing, or moving. (6) In object-oriented design or programming, a service that can be requested at the boundary of an object. Operations include modifying an object or disclosing information about an object.

**operational amplifier** A high-gain amplifier connected to external elements to perform specific operations or functions. (I) (A)

**Operational Assistant** In the AS/400 system, a part of the operating system that provides a set of menus and displays for end users to do commonly performed tasks, such as working with printer output, messages, and batch jobs.

**operational diskette** Synonym for working diskette.

**operational environment** (1) The physical environment; for example, temperature, humidity, and layout. (2) All of the IBM-supplied basic functions and the user programs that can be executed by a store controller to enable the devices in the system to perform specific operations. (3) The collection of IBM-supplied store controller data, user programs, lists, tables, control blocks, and files that reside in a subsystem store controller and control its operation. (4) See also configuration image.

**operational expression** In PL/I, an expression that consists of one or more operations.

**operational key** Synonym for session cryptography key.

**operational mode** See asynchronous balanced mode, asynchronous response mode, normal response mode.

**operational rights** The authority to use an object and to look at its description.

**operational sign** An algebraic sign associated with a numeric data item or a numeric literal that indicates whether the item is positive or negative.

**operational unit (OU) number** In System/36, the number that corresponds to the line connector, located on the back of the system unit, to which a line is attached.

**operation code** (1) A code for representing the operation parts of the machine instructions of a computer. (T) (2) A code used to represent the operations of a computer. (3) In SSP-ICF, a code used by a System/36 application program to request SSP-ICF data management or the subsystem to perform an action; for example, the operation \$SEND asks that data be sent. (4) In RPG, a word or abbreviation, specified in the calculation specifications, that identifies an operation.

**operation code trap** A specific value that replaces the normal operation part of a machine instruction at a particular location to cause an interrupt when that machine instruction is executed. (T)

**operation control language (OCL)** A programming language used to code operation control statements.

**operation control statement** A statement in a job or job step that is used in identifying the job or describing its requirements to the operating system.

**operation decoder** A device that selects one or more control channels according to the operation part of a machine instruction. (A)

**operation expression** An expression containing one or more operators.

**operation mode** The normal working state of a product or system. See also maintenance mode.

**operation part** (1) The part of a machine instruction or microinstruction that specifies the operation to be performed. (T) (2) The part of an instruction that specifies the operation to be performed. Synonymous with function part, operator part. (A) (3) See also implied addressing.

**operations analysis** Synonym for operations research.

DEF084423

**RTAM generation**

[591]

**running open**

**RTAM generation** The process of assembling selected RTAM facilities and link editing them into VS1.

**RTB** Response/throughput bias.

**RTG** Route Table Generator.

**RTM** Realtime monitor.

**RTTY** Radio teletypewriter telecommunications.

**RTV** Real-Time Video.

**RU** Request/response unit.

**rubber-banding** In computer graphics, moving the common ends of a set of straight lines while the other ends remain fixed. (I) (A) See Figure 129.

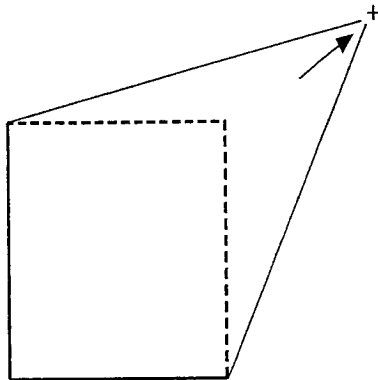


Figure 129. Rubber-Banding

**rubber-band outline** In the AIX operating system, a window with a moveable outline.

**rub-out character** Synonym for delete character.

**RU chain** In SNA, a set of related request/response units (RUs) consecutively transmitted on a particular normal or expedited data flow. The request RU chain is the unit of recovery. If one RU in the chain cannot be processed, the entire chain must be discarded.

**Note:** Each request unit belongs to only one chain, which has a beginning and an end indicated through control bits in request/response headers within the RU chain. Each RU can be designated as first-in-chain (FIC), last-in-chain (LIC), middle-in-chain (MIC), or only-in-chain (OIC). Response units and expedited-flow request units are always sent as only-in-chain.

**rule** A solid or patterned line of any weight, extending horizontally or vertically across a column or row.

**rule-based system** A computer system which performs inferences by applying a set of if then rules to a

set of facts following given procedures. Synonymous with production system. (T)

**rule interpreter** Synonym for inference engine. (T)

**ruler line** A line that indicates where the left and right margins and any tab stops are set. (T)

**run** (1) A performance of one or more jobs. (I) (A) (2) A performance of one or more programs. (I) (A) (3) To cause a program, utility, or other machine function to be performed.

**runaway task** In CICS, a task that does not relinquish control within an interval of time defined by the user.

**RUN disk** The virtual disk that contains the VTAM, NetView, and VM/SNA console support (VSCS) load libraries, program temporary fixes (PTFs) and user-written modifications from the ZAP disk. See BASE disk, DELTA disk, MERGE disk, ZAP disk.

**rundown** In multimedia, an outline of the content of a video program for which a script is inappropriate or impossible, such as an interview.

**run duration** Synonym for running time.

**run file** In the AIX operating system, synonym for load module.

**run-length coding** A technique for compressing data that avoids having to code repeatedly data elements of the same value; instead, the value is coded once, along with the number of times for it to be repeated.

**run list** In VM/SP, a list of virtual machines that are receiving a queue slice on the processing unit. The virtual machine currently executing is called the runuser. When virtual machines are dropped from the run list, replacement is made from the eligible list. See also dispatch list, eligible list.

**running foot** (1) Synonym for footer. (T) (2) A footing that is repeated above the bottom margin area on consecutive pages or on consecutive odd-numbered or even-numbered pages in the text area of the page. Synonymous with footer.

**running heading** (1) Synonym for header. (T) (2) A heading that is repeated below the top margin area on consecutive pages or on consecutive odd-numbered or even-numbered pages in the text area of the page.

**running open** In telegraph applications, a term used to describe a machine connected to an open line or a line without battery (constant space condition). A telegraph receiver under such a condition appears to

DEF084535

*A Merriam-Webster®*

# Webster's Ninth New Collegiate Dictionary

Almost 160,000 entries and 200,000 definitions.

- Entries for words often misused or confused include a clear, authoritative guide to good usage.
- In an exclusive new feature — entries are dated. How old is a word? When was it first used? The answer is here, but in no other American dictionary.
- The newest in the famous Collegiate series, the most widely approved dictionary for home, school and office.



**A GENUINE MERRIAM-WEBSTER**

The name *Webster* alone is no guarantee of excellence. It is used by a number of publishers and may serve mainly to mislead an unwary buyer.

A *Merriam-Webster*® is the registered trademark you should look for when you consider the purchase of dictionaries or other fine reference books. It carries the reputation of a company that has been publishing since 1831 and is your assurance of quality and authority.

Copyright © 1987 by Merriam-Webster Inc.

Philippines Copyright 1987 by Merriam-Webster Inc.

Library of Congress Cataloging in Publication Data  
Main entry under title:

Webster's ninth new collegiate dictionary.

Based on Webster's third new international dictionary.

Includes index.

1. English language—Dictionaries. I. Merriam-Webster Inc.

PE1628.W5638 1987 423 86-23801

ISBN 0-87779-508-8

ISBN 0-87779-509-6 (indexed)

ISBN 0-87779-510-X (deluxe)

Webster's Ninth New Collegiate Dictionary principal copyright 1983

COLLEGIATE trademark Reg. U.S. Pat. Off.

All rights reserved. No part of this book covered by the copyrights hereon may be reproduced or copied in any form or by any means—graphic, electronic, or mechanical, including photocopying, taping, or information storage and retrieval systems—without written permission of the publisher.

Made in the United States of America

2223242526RMcN87

Abbre









**de-scant** \des-'kant, -kənt\ *n* [ME *dyscant*, fr. ONF & ML: ONF *descant*, fr. ML *descantus*, fr. L *dis-* + *cantus* song — more at CHANT] (14c) 1 *a*: a melody or counterpoint sung above the plainsong of the tenor *b*: the art of composing or improvising contrapuntal part music; also: the music so composed or improvised *c*: SOPRANO, TREBLE *d*: a superimposed counterpoint to a simple melody sung typically by some or all of the sopranos 2: discourse or comment on a theme  
**de-scent** \des-'kant, -des-, -dis-\ *v* (15c) 1: to sing or play a descant; broadly: SING 2: COMMENT, DISCOURSE  
**de-scent** \di-'scent\ *vb* [ME *descenden*, fr. OF *descendre*, fr. L *descendere*, fr. *de-* + *scandere* to climb — more at SCAN] *v* (14c) 1: to pass from a higher place or level to a lower one (—ed from the platform) 2: to pass in discussion from what is logically prior or more comprehensive 3 *a*: to come down from a stock or source: DERIVE — usu. used in passive (was —ed from an ancient family) *b*: to pass by inheritance (an heirloom that has —ed in the family) *c*: to pass by transmission (songs —ed from early ballads) *d*: to incline, lead, or extend downward (the road —s to the river) *e*: to swoop or pounce down or make a sudden attack (the plague —ed upon them) 6: to proceed in a sequence or gradation from higher to lower or from more remote to nearer or more recent 7 *a*: to lower oneself in status or dignity: STOOP *b*: to worsen and sink in condition or estimation ~ *w* 1: to pass, move, or climb down or down along 2: to extend down along — **de-scent-bly** \des-'dā-bəl\ *adj*  
**de-scent-ant** or **de-scent-dent** \di-'scent-dənt\ *adj* [MF & L; MF *descendant*, fr. L *descendenti*, *descendens*, prp. of *descendere*] (1572) 1: moving or directed downward 2: proceeding from an ancestor or source  
**descendant** or **descendent** *n* [F & L; F *descendant*, fr. LL *descendenti*, *descendens*, fr. L] (1600) 1: one descended from another or from a common stock 2: one deriving directly from a precursor or prototype  
**des-cender** \di-'sen-dər, -də-, -\ *n* (1802): the part of a lowercase letter (as *p*) that descends below the main body of the letter; also: a letter that has such a part  
**des-cen-sion** \di-'sen-shən\ *n*, *archaic* (14c): DESCENT 2  
**des-cent** \di-'sent\ *n* [ME, fr. MF *descente*, fr. OF *descendre*] (14c) 1 *a*: derivation from an ancestor: BIRTH, LINEAGE (of French ~) *b*: transmission or devolution of an estate by inheritance usu. in the descending line *c*: the fact or process of originating from an ancestral stock *d*: the shaping or development in nature and character by transmission from a source: DERIVATION 2: the act or process of descending 3: a step downward in a scale of gradation; *specif*: one generation in an ancestral line or genealogical scale 4 *a*: an inclination downward: SLOPE *b*: a descending way (as a downgrade or stairway) *c* obs: the lowest part 5 *a*: ATTACK, INVASION *b*: a sudden disconcerting appearance (as for a visit) 6: a downward step (as in station or value): DECLINE (~ of the family to actual poverty)  
**de-scribe** \di-'skrib\ *v* *de-scribed*; **de-scrib-ing** [L *describere*, fr. *de-* + *scribere* to write — more at SCRIBE] (15c) 1: to represent or give an account of in words (~ a picture) 2: to represent by a figure, model, or picture: DELINEATE 3 obs: DISTRIBUTE 4: to trace or traverse the outline of (~ a circle) 5 *archaic*: OBSERVE, PERCEIVE — **de-scrib-able** \di-'skrib-ə-bəl\ *adj* — **de-scriber** *n*  
**de-scrip-tion** \di-'skrip-shən\ *n* [ME *descriptioun*, fr. MF & L; MF *descriptio*, fr. L *descriptio*-, *descriptia*, fr. *descriptus*, pp. of *describere*] (14c) 1 *a*: an act of describing; *specif*: discourse intended to give a mental image of something experienced (as a scene, person, or sensation) *b*: a descriptive statement or account (a fascinating ~ of his adventures) 2: kind or character esp. as determined by salient features (opposed to any tax of so radical a ~) *syn* see TYPE  
**de-scrip-tive** \di-'skrip-tiv\ *adj* (1751) 1: serving to describe (a ~ account) 2: referring to, constituting, or grounded in matters of observation or experience (the ~ basis of science) 3 of a modifier *a*: expressing the quality, kind, or condition of what is denoted by the modified term (hot in "hot water" is a ~ adjective) *b*: NONRESTRICTIVE 4: of, relating to, or dealing with the structure of a language at a particular time usu. with exclusion of historical and comparative data (~ linguistics) — **de-scrip-tive-ly** *adv* — **de-scrip-tive-ness** *n*  
**de-scrip-tor** \di-'skrip-tər\ *n* (1951): something (as an index term) used to identify an item (as a subject or document) esp. in an information retrieval system  
**de-scri-y** \di-'skri\ *v* *de-scried*; **de-scri-y-ing** [ME *descrien*, fr. MF *descrier* to proclaim, decry] (14c) 1 *a*: to catch sight of *b*: to find out: DISCOVER 2 obs: to make known: REVEAL  
**des-try** *n*, obs (1605): discovery or view from afar  
**Des-de-mo-na** \dez-'də-'mə-nə\ *n*: the wife of Othello in Shakespeare's *Othello*  
**de-se-crate** \des-i-'krāt\ *v* *-crated*; *-crat-ing* [*de-* + *-secrete* (as in *consecrate*)] (1677) 1: to violate the sanctity of; PROFANE 2: to treat irreverently or contemptuously often in a way that provokes outrage on the part of others (the kind of shore development ... that has desecrated so many waterfronts — John Fischer) — **de-se-cra-ter** or **de-se-cra-tor** \di-'krāt-ər\ *n*  
**de-se-cra-tion** \des-i-'krā-shən\ *n* (1717): an act or instance of desecrating: the state of being desecrated  
**de-se-gre-gate** \(\)de-'seg-ri-'gāt\ *v* (1952): to eliminate segregation in; *specif*: to free of any law, provision, or practice requiring isolation of the members of a particular race in separate units ~ *w*: to bring about desegregation  
**de-se-gre-ga-tion** \(\)de-'seg-ri-'gā-shən\ *n* (1951) 1: the act or process or an instance of desegregating 2: the state of being desegregated  
**de-se-lect** \di-'sə-'lekt\ *v* (1965): to dismiss (a trainee) from a training program  
**de-sen-si-tize** \(\)de-'sen-(i)-sə-'tīz\ *v* (1898) 1: to make (a sensitized or hypersensitive individual) insensitive or nonreactive to a sensitizing agent 2: to make (a photographic material) less sensitive or completely insensitive to radiation 3: to make emotionally insensitive or callous; *specif*: to extinguish an emotional response (as of fear, anxiety, or guilt) to stimuli that formerly induced it — **de-sen-si-ti-zation** \(\)de-'sen-sai-(ə)-'zā-shən, -sen-sā-'zā-\ *n* — **de-sen-si-tizer** \(\)de-'sen-sai-tīz-ər\ *n*  
**de-ser-er** \dez-ər-\ *n* [ME, fr. OF, fr. LL *desertum*, fr. L neut. of *desertus*, pp. of *deserere* to desert, fr. *de-* + *serere* to join together — more at

SERIES] (13c) 1 a: arid barren land; esp: a tract incapable of supporting any considerable population without an artificial water supply b: an area of water apparently devoid of life 2 *archaic*: a wild uninhabited and uncultivated tract 3: a desolate or forbidding area (lost in a ~ of doubt) — *de-ser-tie* \de-'zər-tik-*əd* *adj*  
*des-ert* \de-'zər-*t* *adj* (13c) 1: desolate and sparsely occupied or unoccupied (~ as ~ island) 2: of or relating to a desert 3 *archaic*: FORSAKEN  
*de-sert* \di-'zər-t *n* [ME *deserte*, fr. OF, fr. fem. of *desert*, pp. of *deservir* to deserve] (13c) 1: the quality or fact of deserving reward or punishment 2: deserved reward or punishment — usu. used in plural (got his just ~) 3: EXCELLENCE WORTH  
*de-sert* \di-'zər-t *vb* [F *désérer*, fr. LL *desertare*, fr. *desertus*] *vi* (1603) 1: to withdraw from or leave usu. without intent to return 2 a: to leave in the lurch (~ a friend in trouble) b: to abandon (military service) without leave ~ vi: to quit one's post, allegiance, or service without leave or justification; esp: to absent oneself from military duty without leave and without intent to return *syn* see ABANDON — *des-ert-er* *n*  
*de-ser-i-fi-ca-tion* \di-'zər-t-ə-'fə-'kā-shən *n* ['desert + -ification (as in saponification)] (1974): the process of becoming arid land or desert (as from land mismanagement or climate change)  
*de-ser-tion* \di-'zər-shən *n* (1591) 1: an act of deserting; esp: the abandonment without consent or legal justification of a person, post, or relationship and the associated duties and obligations (sued for divorce on grounds of ~) 2: a state of being deserted or forsaken  
*desert locust* *n* (1944): a destructive migratory locust (*Schistocerca gregaria*) of southwestern Asia and parts of northern Africa  
*desert soil* *n* (ca. 1938): a soil that develops under sparse shrub vegetation in warm to cool arid climates with a light-colored surface soil usu. underlain by calcareous material and a hardpan layer  
*de-serve* \di-'zər-v *vb* *deserved*; *de-serving* [ME *deserven*, fr. OF *deservir*, fr. L *deservire* to serve zealously, fr. *de-* + *servire* to serve] *vi* (13c): to be worthy of: MERIT (~s another chance) ~ *vi*: to be worthy, fit, or suitable for some reward or requital (have become recognized as they ~ — T. S. Eliot) — *de-server* *n*  
*de-served* \-'zər-vd *adj* (ca. 1552): of, relating to, or being that which one deserves (a ~ reputation) — *de-served-ly* \-'zər-vd-*lē*, -'zər-vd-*lē* *adv* — *de-served-ness* \-'zər-vd-nəs, -'zər-vd-*nəs* *n*  
*de-serv-ing* \-'zər-v-*ing* *n* (14c): DESERT, MERIT (reward the proud according to their ~ — Charles Kingsley)  
*deserving* *adj* (1576): MERITORIOUS, WORTHY: *specif*: meriting financial aid (scholarships for ~ students)  
*de-sex* \('dē-'seks *v* (1911): CASTRATE SPAY  
*de-sex-u-al-ize* \('dē-'seks-(ə)-'wə-'jīz, -'sek-shə-'jīz *vi* (1894) 1: to deprive of sexual characters or power 2: to divest of sexual quality — *de-sex-u-al-iza-tion* \('dē-'seks-(ə)-'wə-lə-'zā-shən, -'sek-shə-'lā-*n*  
*des-ha-bille* \des-ə-'bē(ə)l, -'bīl, -'bē *v* OF DISHABILLE  
*des-i-cant* \des-'i-kənt *n* (1676): a drying agent (as calcium chloride)  
*des-i-cate* \des-'i-kīt *vb* *cat-ed*; *cat-ing* [L *desiccatus*, pp. of *desiccare* to dry up, fr. *de-* + *siccare* to dry, fr. *siccus* dry — more at SACK] *vi* (1575) 1: to dry up 2: to preserve (a food) by drying: DEHYDRATE 3: to drain of emotional or intellectual vitality ~ *vi*: to become dried up — *des-i-ca-tion* \des-'i-kā-shən *n* — *des-i-ca-tive* \des-'i-kāt-iv, di-'sīk-*ət-iv* *adj* — *des-i-ca-tor* \des-'i-kāt-ər *n*  
*de-sid-er-ate* \di-'sīd-ə-'rāt, -'zīd-*v* *vi* *at-ed*; *at-ing* [L *desideratus*, pp. of *desiderare* to desire] (1645): to entertain or express a wish to have or attain — *de-sid-er-a-tion* \di-'sīd-ə-'rā-shən, -'zīd-*n* *n* — *de-sid-er-a-tive* \di-'sīd-ə-'rāt-iv, -'sīd-(ə)-'rāt-iv *adj*  
*de-sid-er-a-tum* \di-'sīd-ə-'rāt-əm, -'zīd-, -'rāt-*n* *n*, pl -*ta* \-tə \[L, neut. of *desideratus*] (1652): something desired as essential  
*de-sign* \di-'zīn *vb* [MF *designer*, fr. L *designare*, fr. *de-* + *signare* to mark, mark out — more at SIGN] *vi* (1548) 1 a: to conceive and plan out in the mind (he ~ed the perfect crime) b: to have as a purpose: INTEND (he ~ed to excel in his studies) c: to devise for a specific function or end (a book ~ed primarily as a college textbook) 2 *archaic*: to indicate with a distinctive mark, sign, or name 3 a: to make a drawing, pattern, or sketch of b: to draw the plans for c: to create, fashion, execute, or construct according to plan: DEVISE, CONTRIVE ~ *vi* 1: to conceive or execute a plan 2: to draw, lay out, or prepare a design — *de-sign-er-ly* \di-'zīn-d-*lē* *adv*  
*design* *n* (1588) 1 a: a particular purpose held in view by an individual or group (he has ambitious ~s for his son) b: deliberate purposive planning (battle was joined... more by accident than ~ — John Buchan) 2: a mental project or scheme in which means to an end are laid down 3 a: a deliberate undercover project or scheme: PLOT b: *pl*: aggressive or evil intent — used with *on* or *against* (he has ~s on the money) 4: a preliminary sketch or outline showing the main features of something to be executed: DELINEATION 5 a: an underlying scheme that governs functioning, developing, or unfolding: PATTERN, MOTIF (the general ~ of the epic) b: a plan or protocol for carrying out or accomplishing something (esp. a scientific experiment); also: the process of preparing this 6: the arrangement of elements or details in a product or work of art 7: a decorative pattern 8: the creative art of executing aesthetic or functional designs *syn* see INTENTION, PLAN  
*des-i-g-nat-e* \de-'zīg-nāt, -nōt *adj* [L *designatus*, pp. of *designare*] (1646): chosen for an office but not yet installed (ambassador ~) — *des-i-g-nate* \-'nāt *vi* *nat-ed*; *nat-ing* (1791) 1: to apart for a specific purpose, office, or duty 2 a: location of (a marker designating the crest of the flo DICATE (any task designated by the employer) c: class (the area we ~ as that of spiritual values)  
*des-i-g-nate* \de-'zīg-nāt, -nōt *adj* [L *designatus*, pp. of *designare*] (1646): chosen for an office but not yet installed (ambassador ~) — *des-i-g-nate* \-'nāt *vi* *nat-ed*; *nat-ing* (1791) 1: to apart for a specific purpose, office, or duty 2 a: location of (a marker designating the crest of the flo DICATE (any task designated by the employer) c: class (the area we ~ as that of spiritual values)  
*des-i-g-nate* \de-'zīg-nāt, -nōt *adj* [L *designatus*, pp. of *designare*] (1646): chosen for an office but not yet installed (ambassador ~) — *des-i-g-nate* \-'nāt *vi* *nat-ed*; *nat-ing* (1791) 1: to apart for a specific purpose, office, or duty 2 a: location of (a marker designating the crest of the flo DICATE (any task designated by the employer) c: class (the area we ~ as that of spiritual values)  
*des-i-g-nate* \de-'zīg-nāt, -nōt *adj* [L *designatus*, pp. of *designare*] (1646): chosen for an office but not yet installed (ambassador ~) — *des-i-g-nate* \-'nāt *vi* *nat-ed*; *nat-ing* (1791) 1: to apart for a specific purpose, office, or duty 2 a: location of (a marker designating the crest of the flo DICATE (any task designated by the employer) c: class (the area we ~ as that of spiritual values)  
*des-i-g-nate* \de-'zīg-nāt, -nōt *adj* [L *designatus*, pp. of *designare*] (1646): chosen for an office but not yet installed (ambassador ~) — *des-i-g-nate* \-'nāt *vi* *nat-ed*; *nat-ing* (1791) 1: to apart for a specific purpose, office, or duty 2 a: location of (a marker designating the crest of the flo DICATE (any task designated by the employer) c: class (the area we ~ as that of spiritual values)  
*des-i-g-nate* \de-'zīg-nāt, -nōt *adj* [L *designatus*, pp. of *designare*] (1646): chosen for an office but not yet installed (ambassador ~) — *des-i-g-nate* \-'nāt *vi* *nat-ed*; *nat-ing* (1791) 1: to apart for a specific purpose, office, or duty 2 a: location of (a marker designating the crest of the flo DICATE (any task designated by the employer) c: class (the area we ~ as that of spiritual values)  
*des-i-g-nate* \de-'zīg-nāt, -nōt *adj* [L *designatus*, pp. of *designare*] (1646): chosen for an office but not yet installed (ambassador ~) — *des-i-g-nate* \-'nāt *vi* *nat-ed*; *nat-ing* (1791) 1: to apart for a specific purpose, office, or duty 2 a: location of (a marker designating the crest of the flo DICATE (any task designated by the employer) c: class (the area we ~ as that of spiritual values)  
*des-i-g-nate* \de-'zīg-nāt, -nōt *adj* [L *designatus*, pp. of *designare*] (1646): chosen for an office but not yet installed (ambassador ~) — *des-i-g-nate* \-'nāt *vi* *nat-ed*; *nat-ing* (1791) 1: to apart for a specific purpose, office, or duty 2 a: location of (a marker designating the crest of the flo DICATE (any task designated by the employer) c: class (the area we ~ as that of spiritual values)  
*des-i-g-nate* \de-'zīg-nāt, -nōt *adj* [L *designatus*, pp. of *designare*] (1646): chosen for an office but not yet installed (ambassador ~) — *des-i-g-nate* \-'nāt *vi* *nat-ed*; *nat-ing* (1791) 1: to apart for a specific purpose, office, or duty 2 a: location of (a marker designating the crest of the flo DICATE (any task designated by the employer) c: class (the area we ~ as that of spiritual values)  
*des-i-g-nate* \de-'zīg-nāt, -nōt *adj* [L *designatus*, pp. of *designare*] (164

|ə| about |ʔ| kitten, F table |ɔr| further |ə| ash  
 |zʊ| out |tʃ| chin |ɛ| bet |i| easy |g| go  
 |ŋ| sing |ɔ| go |ɔ| law |ɔi| boy |θ| thin |  
 |v| yet |z| vision |ə, k, æ, œ, ɪ, ʊ, ʌ, ɪ, ʏ:

deoxy-ribonucleic acid (DNA)

DEF085295



|ə| about    |ɪ| kitten, F table    |ər| further    |æ| ash    |ā| ace    |ā| cot, cart  
 |aʊ| out    |ch| chin    |e| bet    |E| easy    |g| go    |i| hit    |I| ice    |J| job  
 |iŋ| sing    |ō| go    |ó| law    |oɪ| boy    |th| thin    |th| the    |ū| foot    |w| foot  
 |y| yet    |zh| vision    |ā, k̄, , æ, Œ, u, Œ, j| see Guide to Pronunciation

c5yc=t



1074 **sequency • series-wound**

**sequen-cy** \sē-kwən-sē/ *n* [LL *sequentia*] (1818): SEQUENCE 3a. 5  
**sequent** \sē-kwənt/ *adj* [L *sequens*, *sequens*, *pp.*] (1601) 1: CONSECUTIVE SUCCEEDING 2: CONSEQUENT, RESULTANT  
**sequen-tial** \si-kwen-chəl/ *adj* (1854) 1: of, relating to, or arranged in a sequence: SERIAL (~ file systems) 2: following in sequence 3: relating to or based on a method of testing a statistical hypothesis that involves examination of a sequence of samples for each of which the decision is made to accept or reject the hypothesis or to continue sampling — **sequen-tial-ly** \-kwēnch-(ə)-lē/ *adv*  
**se-ques-ter** \si-kwes-tər/ *vi* -tered; -ter-ing \-(tə-)rɪŋ/ [ME *sequestren*, fr. MF *sequester*, fr. LL *sequestare* to surrender for safekeeping, set apart, fr. L *sequester* agent, depository, bailer; akin to L *sequi* to follow] (14c) 1 a: to set apart: SEGREGATE b: to place (property) in custody esp. in sequestration 3: to hold (as a metallic ion) in solution usu. by inclusion in an appropriate coordination complex  
**sequester** *n*, obs (1604): SEPARATION, ISOLATION  
**sequester-ate** \sek-was-ˈtrāt, -sek-; si-kwes-/ *vi* -trat-ed; -trat-ing [LL *sequesteratus*, *pp.* of *sequestrare*] (1513): SEQUESTER  
**se-ques-tration** \sek-was-trā-shən, -sek-/ *n* (15c) 1: the act of sequestering: the state of being sequestered 2 a: a legal writ authorizing a sheriff or commissioner to take into custody the property of a defendant who is in contempt until he complies with the orders of a court b: a deposit whereby a neutral depository agrees to hold property in litigation and to restore it to the party to whom it is adjudged to belong  
**se-ques-trum** \si-kwes-trəm/ *n*, *pl* -trums also -tra \-trə/ [NL, fr. L, legal sequestration; akin to L *sequester* bailer] (1831): a fragment of dead bone detached from adjoining sound bone  
**se-quin** \sē-kwən/ *n* [F, fr. It *zecchino*, fr. *zecca* mint, fr. Ar *sikkah* die, coin] (1582) 1: an old gold coin of Italy and Turkey 2: a small plate of shining metal or plastic used for ornamentation esp. on clothing  
**sequined** or **sequinned** \kwənd/ *adj* (1582): ornamented with or as if with sequins  
**sequi-tar** \sek-wot-ər, -wə-tū(ə)r/ *n* [L, it follows, 3d pers. sing. pres. indic. of *sequi* to follow — more at SUE] (1836): the conclusion of an inference: CONSEQUENCE  
**se-quoia** \si-kwōi-(y)ə/ *n* [NL, genus name, fr. *Sequoia* (George Guess)] (ca. 1866): either of two huge coniferous California trees of the pine family that reach a height of over 300 feet: a: BIG TREE b: REDWOOD 3a  
**sera** *pl* of SERUM  
**serac** \sə-rak, sā-/ *n* [F *serac*, lit., a kind of white cheese, fr. ML *seracum* whey, fr. L *serum* whey — more at SERUM] (1860): a pinnacle, sharp ridge, or block of ice among the crevasses of a glacier  
**seraglio** \sə-ral-(y)ə, -rāl-/ *n*, *pl* -glio [It *seraglio* enclosure, seraglio, partly fr. ML *seraculum* enclosure, bar of a door, bolt, fr. LL *serare* to bolt; partly fr. Turk *saray* palace — more at SEAR] (1588) 1: HAREM 2: a palace of a sultan  
**serai** \sə-rī-/ *n* [Turk & Per; Turk *saray* mansion, palace, fr. Per *sardī* mansion, inn] (1609) 1: CARAVANSARY 2: SERAGLIO 2  
**seral** \sir-əl/ *adj* (1858): of, relating to, or constituting an ecological serie  
**sera-pe** \sə-rāp-ē, -rāp-/ *n* [MexSp *sarape*] (1834): a colorful woolen shawl worn over the shoulders esp. by Mexican men  
**seraph** \ser-əf/ *n*, *pl* ser-a-phim \-ə-fīm/ or seraphs [back-formation fr. *seraphim*] (1667): SERAPHIM 2  
**seraphim** \ser-ə-fīm/ *n*, *pl* [LL *seraphim*, *pl.*, seraphs, fr. Heb *šerāphim*] (bef. 12c) 1: an order of angels — see CELESTIAL HIERARCHY 2 sing, *pl* seraphim: one of the 6-winged angels standing in the presence of God — **seraphic** \sə-rāf-ik/ *adj* — **seraph-ical-ly** \-(ə)-lē/ *adv*  
**Serapis** \sə-rāp-s/ *n* [L, fr. Gk *Sarapis*]: an Egyptian god combining attributes of Osiris and Apis and having a widespread cult throughout Greece and Rome  
**Serb** \sərb/ *n* (Serb *Srb*) (1860) 1: a native or inhabitant of Serbia 2: SERBIAN 2 — *Serb* *adj*  
**Serbo-Croatian** \sər-bē-ōn/ *n* (1862) 1: SERB 2 a: the Serbo-Croatian language as spoken in Serbia b: a literary form of Serbo-Croatian using the Cyrillic alphabet — *Serbian* *adj*  
**Serbo-Croatian** \sər-(b)ō-kro-ʻā-shən/ *n* (1883) 1: the Slavic language of the Serbs and Croats consisting of Serbian written in the Cyrillic alphabet and Croatian written in the Roman alphabet 2: one whose native language is Serbo-Croatian — *Serbo-Croatian* *adj*  
**serē** \sī(ə)r/ *adj* [ME, fr. OE *sear* dry; akin to OHG *sōren* to wither, Gk *haous* dry] (bef. 12c) 1: being dried and withered 2 archaic: THREDBARE  
**serē** *n* [L *series* series] (1939): a series of ecological communities formed in ecological succession  
**serē-nade** \ser-ə-nād/ *n* [F *sérénade*, fr. It *serenata*, fr. *sereno* clear, calm (of weather), fr. L *serenus* serene] (1649) 1 a: a complimentary vocal or instrumental performance; esp: one given outdoors at night for a woman b: a work so performed 2: an instrumental composition in several movements, written for a small ensemble, and midway between the suite and the symphony in style  
**serenade** *vb* -nad-ed; -nad-ing *vt* (1672): to perform a serenade in honor of ~ *vi*: to play a serenade — *serenader* *n*  
**ser-e-na-ta** \ser-ə-nā-tə/ *n* [It, *serenade*] (ca. 1724): an 18th century secular cantata of a dramatic character usu. composed in honor of an individual or event  
**ser-en-dip-i-tous** \ser-ən-dip-ət-əs/ *adj* (1943): obtained or characterized by serendipity (~ discoveries) — *ser-en-dip-i-tous-ly* *adv*  
**ser-en-dip-i-ty** \-dip-ət-ē/ *n* [fr. its possession by the heroes of the Persian fairy tale *The Three Princes of Serendip*] (1754): the faculty of finding valuable or agreeable things not sought for  
**se-re-ne** \sə-rēm/ *adj* [L *serenus*; akin to OHG *serawēn* to become dry, Gk *xēros* dry] (1503) 1: AUGUST — used as part of a title (His Serene Highness) 2: marked by or suggestive of utter calm and unruffled repose or quietude (a ~ smile) 3 a: clear and free of storms or unpleasant change (~ skies) b: shining bright and steady (the moon, ~ in glory — Alexander Pope) *syn* see CALM — *se-re-nely* *adv* — *se-re-ne-ness* \-rēm-nəs/ *n*

**serene** *n* (1644) 1: a serene condition or expanse (as of sky, sea, light) 2: SERENITY, TRANQUILLITY  
**se-ren-i-ty** \sə-rēn-ət-ē/ *n* [ME, fr. MF *serenité*, fr. L *serenitas*, *serenitas* fr. *serenus* serene] (15c): the quality or state of being serene  
**serf** \sərf/ *n* [F, fr. L *servus* slave, servant, *serf* — more at SERVE] (161) a: a member of a servile feudal class bound to the soil and subject to the will of his lord — *serf-age* \sərf-ij/ *n* — *serf-dom* \sərf-dəm, -təm/ *n*  
**serge** \sərdʒ/ *n* [ME *sarge*, fr. MF, fr. (assumed) VL *serica*, fr. L *seric* fem. of *sericus* silken — more at SERICEOUS] (14c): a durable twill fabric having a smooth clear face and a pronounced diagonal rib on the front and the back  
**ser-gean-cy** \sär-jən-sē/ *n* (1670): the function, office, or rank of sergeant  
**ser-geant** \sär-jənt/ *n* [ME, servant, attendant, sergeant, fr. MF *sergent*, fr. L *servient*, *serviens*, *pp.* of *servire* to serve] (14c) 1: SERGEANT AT ARMS 2 obs: an officer who enforces the judgments of court or the commands of one in authority 3: a noncommissioned officer ranking in the army and marine corps above a corporal and below a staff sergeant and in the air force above an airman first class, senior airman and below a staff sergeant; broadly: NONCOMMISSIONED OFFICER 4: an officer in a police force ranking in the U.S. just below captain or sometimes lieutenant and in England just below inspector  
**sergeant at arms** (14c): an officer of an organization (as a legislative body or court of law) who preserves order and executes commands  
**sergeant first class** *n* (1948): a noncommissioned officer in the arm ranking above a staff sergeant and below a master sergeant  
**sergeant fish** *n* (ca. 1883) 1: COBIA 2: SNOOK 1  
**sergeant major** *n*, *pl* sergeants major or sergeant majors (1802) 1: noncommissioned officer in the army, air force, or marine corps serving as chief administrative assistant in a headquarters 2: a noncommissioned officer in the marine corps ranking above a first sergeant 3: bluish green to yellow percoid fish (*Abudefduf saxatilis*) with blue vertical stripes on the sides that is widely distributed in the western tropical Atlantic ocean  
**sergeant major of the army** (1966): the ranking noncommissioned officer of the army serving as adviser to the chief of staff  
**sergeant major of the marine corps** (ca. 1971): the ranking noncommissioned officer of the marine corps serving as adviser to the commandant  
**ser-gean-ty** \sär-jənt-ē/ *n*, *pl* -geant-ies [ME *sergeantie*, fr. MF *sergent*, fr. *sergent* sergeant] (15c): any of numerous feudal services of a personal nature by which an estate is held of the king or other lord distinct from military tenure and from socage tenure  
**ser-ging** \sər-jɪŋ/ *n* [serge] (ca. 1909): the process of overcasting the raw edges of a piece of fabric (as a carpet) to prevent raveling  
**se-ri-al** \sir-ē-əl/ *adj* (1841) 1: of, relating to, consisting of, or arranged in a series, rank, or row (~ order) 2: appearing in successive parts or numbers (a ~ story) 3: belonging to a series maturing periodically rather than on a single date (~ bonds) 4: of, relating to, being music based on a series of tones in an arbitrary but fixed pattern without regard for traditional tonality (~ technique) — *se-ri-al-ly* \-ə-lē/ *adv*  
**serial** *n* (1846) 1 a: a work appearing (as in a magazine or on television) in parts at intervals b: one part of a serial work: INSTALLMENT 2: a publication (as a newspaper or journal) issued as one of a consecutively numbered and indefinitely continued series  
**se-ri-al-ism** \sir-ē-ə-liz-əm/ *n* (1963): serial music; also: the theory practice of composing serial music  
**se-ri-al-ist** \sir-ē-əl-ist/ *n* (1846) 1: a writer of serials 2: a composer of serial music  
**se-ri-al-ize** \sir-ē-əl-iz/ *vi* -ized; -iz-ing (1892): to arrange or publish in serial form — *se-ri-al-iza-tion* \sir-ē-əl-ə-lə-zā-shən/ *n*  
**serial number** *n* (1896): a number identifying place in a series and used as a means of identification  
**se-ri-ate** \sir-ē-āt, -ē-ət/ *adj* [(assumed) NL *seriatus*, fr. L *series*] (184) a: arranged in a series or succession — *se-ri-ate-ly* *adv*  
**se-ri-ate** \sir-ē-āt/ *vi* -at-ed; -at-ing (1872): to arrange in a series  
**se-ri-at-um** \sir-ē-āt-əm, -at-əd/ *n* [ML, fr. L *series*] (1680): in a series  
**seriatim** \sir-ē-āt-əm/ *adv* following serially  
**se-ri-ce-ous** \sə-rish-əs/ *adj* [LL *sericeus* silken, fr. L *sericum* silk; ment. silk, fr. neut. of *sericus* silken, fr. Gk *serikos*, fr. *Seres*, an eastern Asian people producing silk in ancient times] (ca. 1777): finely pubescent (~ leaf)  
**ser-i-cin** \ser-ə-sən/ *n* [TSV, fr. L *sericum* silk] (ca. 1868): a gelatino protein that cements the two fibroin filaments in a silk fiber  
**ser-i-cul-ture** \ser-ə-kəl-chər/ *n* [L *sericum* silk + E *culture*] (185) a: the production of raw silk by raising silkworms — *ser-i-cul-tur-ist* \-kəlch-(ə)-rəl/ *adj* — *ser-i-cul-tur-ist* \-rəst/ *n*  
**se-ries** \si(ə)r-(j)ēz/ *n*, *pl* series often attrib [L, fr. *serere* to join, lit. together; akin to L *sorti*, *sort* lot, Gk *eirein* to string together, *horm* chain, necklace] (1611) 1 a: a number of things or events of the same class coming one after another in spatial or temporal succession (a concert ~) (the hall opened into a ~ of small rooms) b: a set regularly presented television programs each of which is complete itself 2: the indicated sum of a usu. infinite sequence of numbers a: the coins or currency of a particular country and period b: group of postage stamps in different denominations 4: a succession of volumes or issues published with related subjects or authors, similar format and price, or continuous numbering 5: a division of rock formations that is smaller than a system and comprises rocks deposited during an epoch 6: a group of chemical compounds related in composition and structure 7: an arrangement of the parts of or element in an electric circuit whereby the whole current passes through each part or element without branching 8: a set of vowels connected abiaut (as *i*, *e*, *u* in *ring*, *rang*, *rung*) 9 a: a number of games (as baseball) played usu. on consecutive days between two teams (in tot for a 3-game ~) b: WORLD SERIES 10: a group of successive consecutive sentence elements joined together (an *a*, *b*, and *c* ~) 11: SC SERIES 12: three consecutive games in bowling — *in series*: in a serial arrangement  
**series winding** *n* (ca. 1909): a winding in which the armature coil and the field-magnet coil are in series with the external circuit — *se-ri-e wound* \sir-ē-z-ˈwaund/ *adj*

## 1132 special • spectrofluorometric

- eminence or preference; **SPECIFIC** implies a quality or character distinguishing a kind or a species; **PARTICULAR** stresses the distinctness of something as an individual; **INDIVIDUAL** implies unequivocal reference to one of a class or group.
- special** *n* (ca. 1909) 1: something (as a television program) that is not part of a regular series 2: one that is used for a special service or occasion (caught the commuter ~ to work)
- special assessment** *n* (1875): a specific tax levied on private property to meet the cost of public improvements that enhance the value of the property
- special delivery** *n* (1886): expedited messenger delivery of mail matter for an extra fee
- special district** *n* (1950): a political subdivision of a state established to provide a single public service (as water supply or sanitation) within a specific geographical area
- special drawing rights** *n* (1967): a means of exchange used by governments to settle their international indebtedness
- special effects** *n pl* (1944): visual or sound effects introduced into a motion picture or a taped television production during laboratory processing
- Special Forces** *n pl* (1962): a branch of the army composed of men specially trained in guerrilla warfare
- special handling** *n* (1928): the handling of parcel-post or fourth-class mail as first-class but not as special-delivery matter for an extra postal fee
- special interest** *n* (1910): a person or group seeking to influence legislative or government policy to further often narrowly defined interests; esp: **LOBBY**
- special-ism** *\spesh-ə-liz-əm* *n* (1856) 1: specialization in an occupation or branch of learning 2: a field of specialization: **SPECIALTY**
- specialist** *\spesh-(ə-)list* *n* (1856) 1: one who devotes himself to a special occupation or branch of learning 2: any of four enlisted ranks in the army corresponding to the grades of corporal through sergeant first class — **specialist or special-is-tic** *\spesh-ə-lis-tik* *adj*
- special-ity** *\spesh-ē-əl-ə-tē* *n, pl -ties* (15c) 1: a special mark or quality 2: a special object or class of objects 3: a special aptitude or skill b: a particular occupation or branch of learning
- special-iza-tion** *\spesh-(ə-)lā-zā-shən* *n* (1843) 1: a making or becoming specialized 2: a: structural adaptation of a body part to a particular function or of an organism for life in a particular environment b: a body part or an organism adapted by specialization
- special-ize** *\spesh-ə-līz* *vb -ized; -izing* *vi* (1613) 1: to make particular mention of: **PARTICULARIZE** 2: to apply or direct to a specific end or use (*specialized his study*) ~ *vi* 1: to concentrate one's efforts in a special activity or field 2: to undergo specialization; esp: to change adaptively (the sloth became highly *specialized* in the course of evolution)
- specialized** *adj* (1853) 1: designed or fitted for one particular purpose or occupation (~ personnel) 2: characterized by or exhibiting biological specialization; esp: highly differentiated esp. in a particular direction or for a particular end
- special jury** *n* (1730): a jury chosen by the court on request from a list of better educated or presumably more intelligent prospective jurors for a case involving complicated issues of fact or serious felonies — called also *blue-ribbon jury*
- special pleading** *n* (1684) 1: the allegation of special or new matter to offset the effect of matter pleaded by the opposite side and admitted, as distinguished from a direct denial of the matter pleaded 2: misleading argument that presents one point or phase as if it covered the entire question at issue
- special theory of relativity** (1924): **RELATIVITY 3a**
- specialty** *\spesh-əl-ē* *n, pl -ties* [ME *specialite*, fr. MF *especialité*, fr. LL *specialiat-, specialitas*, fr. L *specialis* special] (14c) 1: a distinctive mark or quality 2: a special object or class of objects; as (1): a legal agreement embodied in a sealed instrument (2): a product of a special kind or of special excellence (fried chicken was father's ~) b: the state of being special, distinctive, or peculiar 3: something in which one specializes
- speci-a-tion** *\spē-(h)ē-ā-shən* *n* (ca. 1900): the process of biological species formation — **speci-ate** *\spē-(h)ē-āt* *vi* — **speci-a-tional** *\spē-(h)ē-ā-shən-l* *adj*
- specie** *\spē-shē, -sē* *n* [fr. *specie*, fr. L. in kind] (1617): money in coin — in *specie*: in the same or like form or kind (ready to return insult in *specie*); also: in coin
- specie** *n* [back-formation fr. *species* (taken as a pl.)] *subst* (1711): **SPECIES**
- species** *\spē-(h)shēz, -(h)sē* *n, pl species* [L. appearance, kind, species — more at *spy*] (1551) 1: a class of individuals having common attributes and designated by a common name; *specif*: a logical division of a genus or more comprehensive class b: **KIND, SORT** c: the human race: human beings — often used with the (survival of the ~ in the nuclear age) d (1): a category of biological classification ranking immediately below the genus or subgenus, comprising related organisms or populations potentially capable of interbreeding, and being designated by a binomial that consists of the name of a genus followed by a Latin or latinized uncapitalized noun or adjective agreeing grammatically with the genus name (2): an individual or kind belonging to a biological species e: a particular kind of atomic nucleus, atom, molecule, or ion 2: the consecrated eucharistic elements of the Roman Catholic or Eastern Orthodox Eucharist 3: a mental image; also: a sensible object b: an object of thought correlative with a natural object
- species** *adj* (1899): belonging to a biological species as distinguished from a horticultural variety (a ~ rose)
- species-ism** *\spē-shē-ziz-əm, -sē-* *n* [*species* + *-ism* (as in *racism*)] (1973): prejudice or discrimination based on species; esp: discrimination against animals
- spec-ific** *\spi-'sif-ik* *adj* [LL *specificus*, fr. L *species*] (1631) 1: constituting or falling into a specifiable category b: sharing or being those properties of something that allow it to be referred to a particular category 2: a: restricted to a particular individual, situation, relation, or effect (a disease ~ to horses) b: exerting a distinctive influence (as on a body part or a disease) (~ antibodies) 3: free from ambiguity: **ACCURATE** (a ~ statement of faith) 4: of, relating to, or constituting a species and esp. a biologic species 5: a: being any of various arbitrary physical constants and esp. one relating a quantitative attribute to unit mass, volume, or area b: imposed at a fixed rate per unit (as of weight or count) (~ import duties) — compare *AD VALOREM* *syn* see **SPECIAL EXPLICIT** — **speci-fi-cally** *\i-'k(ə-)lē* *adv*
- specific** *n* (1661) 1: something peculiarly adapted to a purpose or use b: a drug or remedy having a specific mitigating effect on a disease 2: a: a characteristic quality or trait b: **DETAILS, PARTICULARS** — usu. used in pl. (haggling over the legal and financial ~s of independence — *Time*) c *pl*: **SPECIFICATION 2a**
- speci-fi-ca-tion** *\spes-(ə-)fā-'kā-shən* *n* (1615) 1: the act or process of specifying 2: a: a detailed precise presentation of something or of a plan or proposal for something — usu. used in pl. b: a statement of legal particulars (as of charges or of contract terms); also: a single item of such statement c: a written description of an invention for which a patent is sought
- specific epithet** *n* (1947): the Latin or latinized noun or adjective that follows the genus name in a taxonomic binomial
- specific gravity** *n* (1666): the ratio of the density of a substance to the density of some substance (as pure water or hydrogen) taken as a standard when both densities are obtained by weighing in air
- specific heat** *n* (1832) 1: the ratio of the quantity of heat required to raise the temperature of a body one degree to that required to raise the temperature of an equal mass of water one degree 2: the heat in calories required to raise the temperature of one gram of a substance one degree centigrade
- specific impulse** *n* (1947): the thrust produced per unit rate of consumption of the propellant that is usu. expressed in pounds of thrust per pound of propellant used per second and that is a measure of the efficiency of a rocket engine
- speci-fi-ci-ty** *\spes-ə-'fis-ə-tē* *n* (1876): the quality or condition of being specific; as a: the condition of being peculiar to a particular individual or group of organisms (host ~ of a parasite) b: the condition of participating in or catalyzing only one or a few chemical reactions (the ~ of an enzyme)
- specific performance** *n* (1873) 1: the performance of a legal contract strictly or substantially according to its terms 2: an equitable remedy enjoining specific performance
- speci-ify** *\spes-ə-'fi* *vt -fied; -fying* [ME *specifier*, fr. MF *specifier*, fr. LL *specificare*, fr. *specificus*] (14c) 1: to name or state explicitly or in detail 2: to include as an item in a specification — **speci-fi-able** *\fi-'ə-bəl* *adj* — **speci-fi-er** *\fi-'ə-r* *n*
- speci-men** *\spes-(ə-)mən* *n* [L. fr. *specere* to look at, look — more at *spy*] (1610) 1: an item or part typical of a group or whole 2: a: something that obviously belongs to a particular category but is noticed by reason of an individual distinguishing characteristic b: **PERSON, INDIVIDUAL** (he's a tough ~) *syn* see **INSTANCE**
- speci-os-ity** *\spē-shē-'is-ə-tē* *n* (1608): the quality or state of being specious
- specious** *\spē-shəs* *adj* [ME, fr. L *speciosus* beautiful, plausible, fr. *specer*] (15c) 1: **SHOWY** 2: having deceptive attraction or allure 3: having a false look of truth or genuineness: **SOPHISTIC** — **speciously** *adv* — **specious-ness** *n*
- speck** *\spek* *n* [ME *specke*, fr. OE *specca*] (bef. 12c) 1: a small discoloration or spot esp. from stain or decay 2: a very small amount 3: something marked or marred with specks — **specked** *\spek-t* *adj*
- speck** *vt* (1580): to produce specks on or in
- speck-le** *\spek-əl* *n* [ME, akin to OE *specca*] (15c): a little speck (as of color)
- speckle** *vt* **speck-led; speck-ling** *\(ə-)līŋ* (ca. 1570) 1: to mark with speckles 2: to be distributed in or on like speckles
- speckled perch** *n* (1888): **BLACK CRAPPIE**
- speckled trout** *n* (1805) 1: **BROOK TROUT** 2: **SPOTTED SEA TROUT**
- specs** *\speks* *n pl* [contr. of *spectacles*] (1807): **EYEGLASSES**
- specs** *n pl* [by contr.] (1942): **SPECIFICATIONS**
- spect-a-cle** *\spek-ti-kəl* *also -tik-əl* *n* [ME, fr. MF, fr. L *spectaculum*, fr. *specere* to watch, fr. *spectus*, pp. of *specere* to look, look at — more at *spy*] (14c) 1: a: something exhibited to view as unusual, notable, or entertaining; esp: an eye-catching or dramatic public display b: an object of curiosity or contempt (made a ~ of herself) 2: **GLASSES** 3: something (as natural markings on an animal) suggesting a pair of glasses
- spect-a-cled** *\-ti-kəld, -tik-əld* *adj* (1607) 1: having or wearing spectacles 2: having markings suggesting a pair of spectacles (a ~ alligator)
- spect-a-cu-lar** *\spek-'tak-yə-lər, spēk-əl* *adj* [L *spectaculum*] (1682): of, relating to, or constituting a spectacle: **STRIKING, SENSATIONAL** (a ~ display of fireworks) — **spect-a-cu-lar-ly** *adv*
- spectacular** *n* (1890): something that is spectacular
- spect-a-tor** *\spek-'tāt-ər, spēk-əl* *n* [L. fr. *spectator*, pp. of *specere* to watch] (1586): one who looks on or watches — **spectator** *adj*
- specter or spectre** *\spek-tər* *n* [F *spectre*, fr. L *spectrum* appearance, specter, fr. *specere* to look, look at — more at *spy*] (1605) 1: a visible disembodied spirit: **GHOST** 2: something that haunts or perturbs the mind: **PHANTASM** (the ~ of hunger)
- spec-ti-no-my-cin** *\spek-tō-nō-'mīs-n* *n* [NL, fr. *spectabilis* + *-in* + *-o-* + *-mycin*] (ca. 1964): a white crystalline broad-spectrum antibiotic  $C_{14}H_{21}N_7O_6$  produced by a bacterium (*Streptomyces spectabilis*) that is used clinically esp. in the form of its hydrochloride to treat gonorrhea
- spect-tral** *\spek-trəl* *adj* (1815) 1: of, relating to, or suggesting a specter: **GHOSTLY** 2: of, relating to, or made by a spectrum — **spect-tral-ly** *adv*
- spectral line** *n* (1902): one of a series of linear images of the narrow slit of a spectrograph or similar instrument corresponding to a component of the spectrum of the radiation emitted by a particular source
- spectro-comb form** [NL *spectrum*]: spectrum (*spectroscope*)
- spectro-flu-o-m-e-t-er** *\spek-(trō-, flu-(ə)-)r-əm-ē-t-ər* *also* **spectro-flu-o-rim-e-t-er** *\-im-əl* *n* [*spectr-* + *fluorimeter*] (1962): a device for measuring and recording fluorescence spectra — **spectro-flu-o-ro-met-ric**

## CAD Systems for IC Design

MARVIN E. DANIEL AND CHARLES W. GWYN, SENIOR MEMBER, IEEE

**Abstract**—As integrated circuit (IC) complexities increase, many existing computer-aided design (CAD) methods must be replaced with an integrated design system to support very large scale integrated (VLSI) circuit and system design. The framework for a hierarchical CAD system is described. The system supports both functional and physical design from initial specification and system synthesis to simulation, mask layout, verification, and documentation. The system is being implemented in phases on a DECSystem 20 computer network and will support evolutionary changes as new technologies are developed and design strategies defined.

### INTRODUCTION

COMPUTER-AIDED DESIGN (CAD) of integrated circuits (IC's) has had various interpretations as a function of time and definition source. These interpretations range from use of simple, interactive graphics and digitizing systems to individual programs used for circuit or logic simulation, mask layout, and data manipulation or reformatting. In many instances, the word "aided" is deemphasized to imply nearly automatic design. Within the context used in this paper, CAD refers to a collection of software tools to provide the designer with design assistance during each phase of the design. Although many decisions are made by the software during the design process, important decisions are the designer's responsibility. The computer aids or tools provide the designer with a rapid and orderly method for consolidating and evaluating design ideas and relieve the designer of numerous routine and mechanistic design steps.

### Background

A brief review of some of the techniques and terminology that evolved into today's CAD is instructive to viewing the collection of tools available today. CAD had its origins in the mid-to-late 1950's. With the advent of high-speed digital computers, some of the pioneering work of Kron [1], [2] was applied to the simultaneous solution of network equations. This formulation was used, in part, by computer codes such as NET-1 [3] (Network Analysis Program) and ECAP [4] (Electronic Circuit Analysis Program). From a physical point of view, solution of the network problem predicts the behavior of a system in terms of the element characteristics and interconnections. Viewed as a mathematical problem, the properties of a topological structure (linear graph) and a superimposed algebraic structure (interrelations of the nodes, branches, and meshes of the graph) are determined.

This early work of solving steady-state and transient network problems was the genesis of circuit simulation and was designated CAD. By the early 1970's, the growing need to simulate large circuits and thus obtain the detailed solution of a large number of partial differential equations forced the development of optimization methods and higher levels of abstraction to represent physical systems. Whereas very detailed models are used to simulate the behavior of individual transistors, more abstract representations are used for timing and logic simulation. A timing simulator uses only current-voltage tables for transistor models; capacitive loading, and circuit connectivity to determine the signal waveforms at each circuit node. Logic or gate-level simulation solves the equivalent Boolean equations with delay elements inserted between gates to account for signal timing.

The use of computer aids in the layout of IC masks essentially proceeded along two approaches: interactive graphic systems and automatic layout based on standard cells. Early interactive graphic systems provided a method for capturing design by recording coordinate information by manually digitizing and editing data. Methods for superimposing multiple layers, scaling, enlarging, contrasting, and reviewing the results were expanded to provide fast, sophisticated drawing commands for constructing, editing, and reproducing complex figures; performing dimensional tolerance checks; selective erase, expand, move, and merge; symbolic input; pattern generation, etc.

Automatic layout methods have classically relied on a library of circuit components or cells in the form of mask geometries defining logic gates. Most software required cells with standard heights and varying width. The layout software, placing cells in rows, attempts to optimize the cell position in the row and interconnects the cells in wiring channels between the rows. The automatic standard cell layout programs have evolved from simple linear cell placement in a single row which was subsequently folded to fit a square area [5]. More complex placement in two dimensions. The early standard cell layout aids supported the use of single entry (connections on one side of the cell) cells placed in the row in a back-to-back configuration. Power was distributed to the cells through a common connection on the backside of the cell. Newer standard cell layout programs support cells containing terminals on both sides of the cell. Instead of placing cells in a back-to-back configuration, each cell row contains a linear placement of the cells [6], [7] and is separated by wiring channels.

The early use of CAD for physical design verification consisted of performing simple design rule checks for width and spacing violations on mask artwork files. Functional integrity was verified through circuit simulations.

Manuscript received April 10, 1981; revised September 8, 1981. This work was supported by the U.S. Department of Energy.  
The authors are with Sandia National Laboratories, Integrated Circuit Design, Dept. 2110, Albuquerque, NM 87185.



DANIEL AND GWYN: CAD SYSTEMS FOR IC DESIGN

*Initial Sandia Design Aids*

After the initial decision to establish a CAD capability for integrated circuits at Sandia, software was obtained from universities and private industry wherever possible to provide a basic capability. For example, the SPICE [8] circuit analysis code was obtained from the University of California at Berkeley, and the PR2D [6] standard cell layout program for metal gate cells was obtained from RCA. Since Sandia had previously developed a simple logic analysis capability, this work was accelerated to provide gate-level simulation. Software was developed to postprocess the mask layout data to pattern generator formats for mask generation and plot file information for the Xynetics flatbed plotter. A software package was purchased from Systems, Science, and Software, Inc., and installed on an existing interactive graphic system. In addition, translator subroutines were written to convert design information data formats to minimize the manual data translation and to allow the design information to be added only once in the system. The above process is oversimplified, since a substantial commitment of staff was required to: (1) understand the acquired software and debug and correct errors, (2) perform modifications to support the software on Control Data computers, (3) identify and develop required translator software, and (4) develop additional design aids. These problems and the subsequent software modifications required a large manpower investment, since most of the software was not documented, had evolved over many years with several authors, used nonstandard Fortran, and was unstructured.

*CAD System Requirements*

Although the initial design aids provided a valuable capability for simple metal gate CMOS custom IC design, many deficiencies were identified. These deficiencies coupled with the need to support new technologies with shrinking design rules and thus rapidly increasing circuit complexities, new design needs, and changing design objectives, required a new approach to the development of aids for IC design.

Several general objectives for new aids were identified. The aids must (1) be user oriented, (2) use modular software, (3) be evolutionary to meet changing design needs, and (4) be integrated into a complete design system rather than exist as an independent collection of disjoint tools.

Computer aids must support both functional and physical design. Functional design aids include synthesis, verification, simulation, and testing at architecture, system, logic, circuit, device, and process levels. Physical design aids support partitioning, layout, and topological analysis at all design levels. Functionality, testability, and physical design must be considered in parallel throughout the design process.

All CAD software must be as technology independent as possible and support various levels of designer sophistication from inexperienced first-time users to state-of-the-art system designers. To meet these goals, the interaction between individual programs, data base, and design engineer must be through a single, consistent interface. This interface should support monitoring the progress of a design, supply options at any stage in the design process, and assist in design document-

TABLE I

Complexity	Examples
VLSI (~10 <sup>5</sup> devices)	Microcomputers, cryptographic circuits, ROMs, RAMs
LSI (~10 <sup>4</sup> devices)	Microprocessors, A/D & D/A converters, ALUs, FFT circuits, ROMs, RAMs
MSI (~10 <sup>3</sup> devices)	Address, complex gates, multiplexers, ROMs, RAMs
SSI (~100 devices)	AND, OR, NAND, NOR gates, buffers, memory cells
Primitive elements	Transistors, resistors, capacitors

tation and maintenance.

A system design language—or Hardware Description Language (HDL)—must be available for describing all levels of system behavior and structure. Organized in a hierarchical manner, the language should support functional and physical descriptions and the relationships among entities.

Synthesis aids must facilitate the addition of sufficient detail to generate a complete system description. Design verification software monitors internal consistency and completeness of system specifications. Final system specifications should be retained in a dynamic data base providing files in the proper format for input to each of the design aids and for documentation.

A complete CAD system satisfying the above requirements has been designed and implementation is in progress. The basic system description has been divided into three sections. The first section describes design flow and the concept of top-down design with bottom-up implementation. A hierarchical design approach is used with appropriate merging of levels to accomplish the design of VLSI systems. Requirements for specific computer aids are outlined in the second section. The final section describes the implementation philosophy, computer hardware, and present software development status.

*HIERARCHICAL DESIGN*

The CAD system supports a number of functions at each level in the design hierarchy. Design proceeds in a top-down sequence with bottom-up detailed implementation and addresses both functional and physical problems at each level.

Architecture, the top level of the design hierarchy, contains elements for a broad functional system description, requirements for interfacing units specifying performance and compatibility, and methods for partitioning the system into major functional blocks such as processors, memory, and I/O. The architectural specification can be expanded at the system level to generate a more detailed description consisting of register transfer level subsystem functions.

At the logic level, system functions are defined as interconnections of fundamental gates or modules. Logic modules can be further decomposed into circuit primitives (e.g., transistors, resistors, capacitors). Finally, in order to construct circuit elements and determine their behavior, the physical implementation and process technology may be considered.

To support a variety of technologies in a hierarchical design structure, a data base consisting of a library of elements of varying sophistication must be maintained. A distinct library must exist for each technology used. Typically, the hierarchi-

DEF011952

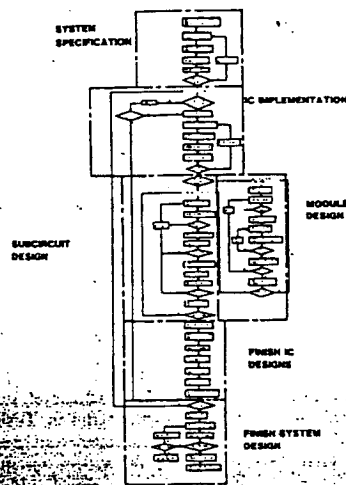


Fig. 1. System design sequence.

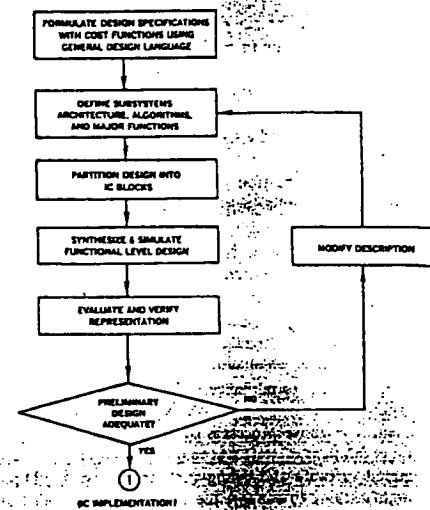


Fig. 2. System specification and partitioning.

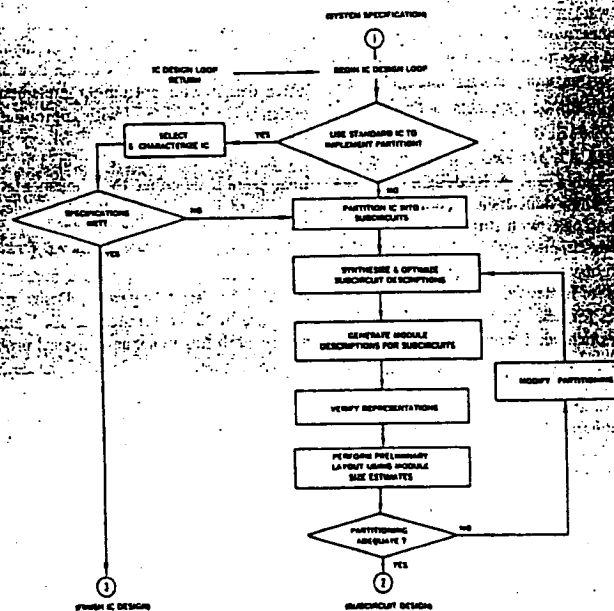


Fig. 3. IC design planning.

cal library will contain circuits and subcircuits at various levels of complexity (VLSI, LSI, MSI, SSI, circuit primitives and/or discrete devices) as defined in Table I.

Electronic system design using computer aids can be divided into six major design sections: (1) system specification and

partitioning, (2) IC design planning and initial implementation, (3) subcircuit design, (4) module design, (5) completion of individual IC design, and (6) completion of system design. A design flow diagram summarizing these major functions is shown in Fig. 1.



RY 19 DANIEL AND GWYN: CAD SYSTEMS FOR IC DESIGN

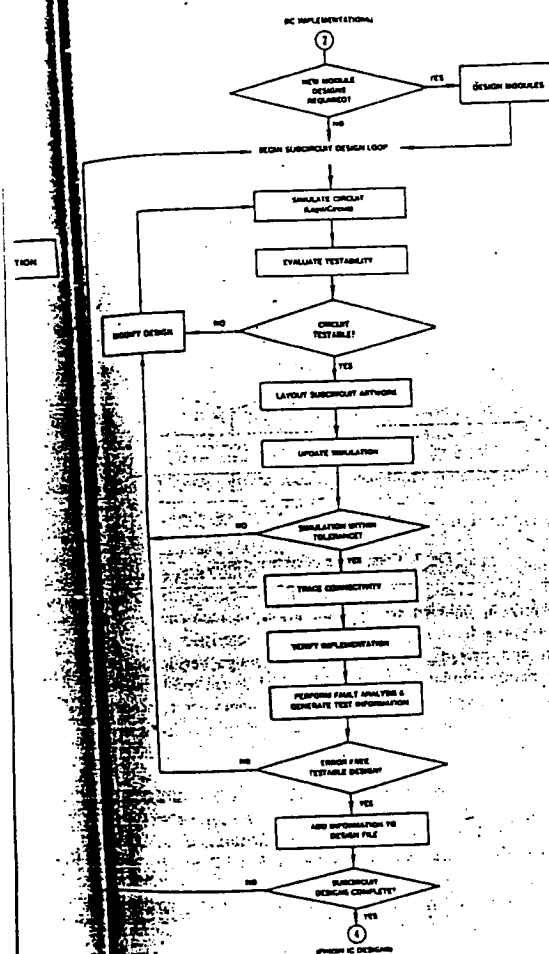


Fig. 4. Subcircuit design.

The flow diagram in Fig. 1 is subdivided and enlarged to show the individual design steps in Figs. 2-7. The first major step consists of developing system specifications and partitioning guidelines (Fig. 2). After developing general specifications, systems, architecture, algorithms, and major functions are defined. Each subsystem is partitioned into IC blocks which are synthesized and simulated at the system design level. This preliminary design is evaluated and modified if necessary by repeating the above steps.

The next major step in the design is implementation of each subsystem based on the system specification (Fig. 3). For a general electronic system, the first decision to be made is whether custom or commercial circuits will be used. If a characterized commercial circuit is available in the library, the circuit is se-

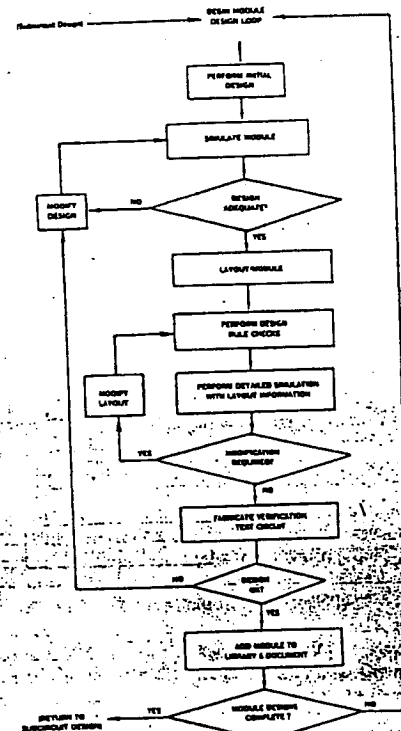


Fig. 5. Module design.

lected. If the required commercial circuit is not in the library, the specifications must be obtained and compared with the system specification. If the system specifications are met, the commercial circuit can be used; if not, a custom circuit must be designed.

The first step in designing a custom IC is partitioning the IC into subcircuits. Each subcircuit is synthesized and optimized. Module descriptions for each subcircuit are developed and verified, and a preliminary layout using the estimated module sizes is performed. At this point, a decision can be made concerning the adequacy of the partitioning. If the preliminary layout does not conform to the size restrictions for the circuit or if during the circuit verification, signal-delay paths are longer than desired, the partitioning must be modified and the circuit design repeated.

If partitioning is acceptable, the subcircuit design step can be initiated (Fig. 4). For each subcircuit design, a decision must be made concerning the adequacy of modules in the library for implementing the design; new modules must be designed and characterized as required (Fig. 5).

During subcircuit implementation, each circuit can be simulated at various individual or combined simulation levels, the testability evaluated, and the artwork generated. After artwork

DEF011954

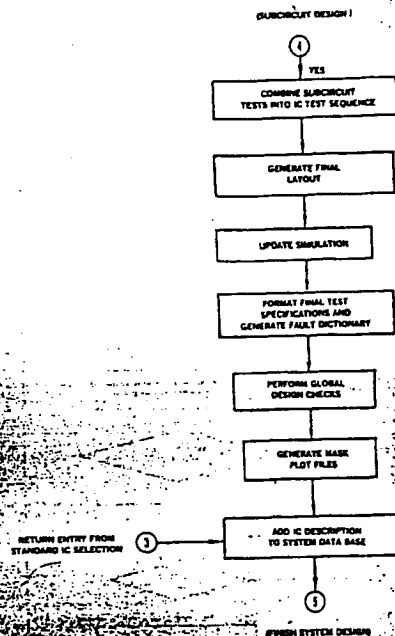


Fig. 6. Completion of IC design.

work design, the simulation net list must be updated to include circuit parasitics associated with the specific layout. This simulation also provides information concerning circuit output drive and power required for subcircuit operation. If the simulation is within tolerance, static design verification is initiated; connectivity is traced; and the layout is checked for conformance to design rules. The layout and associated simulation characteristics are compared with the initial circuit specifications. Finally, test information is generated for the subcircuit.

After each subcircuit design has been completed and verified, the circuit design information is added to a design file in the data base. The next major step consists of completing the IC design by combining subcircuit design information (Fig. 6). The IC layout is assembled by placing and interconnecting subcircuit layouts. The simulation is updated to assure correct circuit operation, test specifications are generated, and a fault dictionary is developed for system diagnosis. Physical design verification, including global rule checks and connectivity checks, is performed, and the mask information for each IC is generated. Finally, design information for each IC is added to the system data base.

The last major step in the design sequence consists of completing the system design (Fig. 7). After selecting commercial circuits or designing custom IC's, layouts are performed for the system circuit boards. After the printed- or hybrid-circuit board layouts have been completed, the entire system is simu-

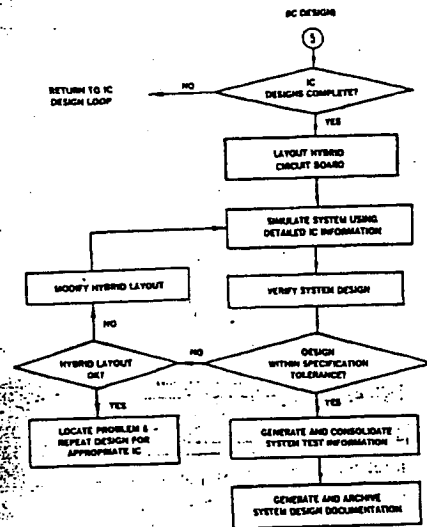


Fig. 7. Completion of system design.

lated, using detailed design information for each circuit. The simulation determines the electrical characteristics of the system and provides a final functional design verification by comparing simulation results with initial design specifications. Test sequences for the individual IC's are combined to produce system tests at the printed-circuit board or hybrid-circuit level. During the final step, system design documentation is generated and archived. Generating system documentation consists of obtaining information from the design data base and consolidating it into the appropriate format. In addition to the mask plot files and test-sequence information, the documentation includes system performance information with tolerance specifications for acceptance of the final system. The complete CAD system for IC design will include a number of computer programs. These aids can be categorized as follows: design specification and partitioning, system and circuit synthesis, system partitioning, simulation at various levels, IC mask layout, design verification, testability evaluation, test sequence generation, data base, and design documentation.

#### SPECIFIC COMPUTER AIDS REQUIREMENTS

In addition to a number of specific computer programs for supporting a hierarchical design, an extensive data base and support software for monitoring the design flow and communication are required. Requirements for specific programs and interfaces are outlined below.

#### Design Executive

Currently, most design automation programs exist as independent entities with idiosyncratic user interfaces and incompatible I/O structures. A Design Executive can provide a single

DEF011955

JANUARY 1988 DANIEL AND GWYN: CAD SYSTEMS FOR IC DESIGN 7

consistent interface between a user and the complete set of computer aids available on the CAD system.

The Design Executive supports the complex CAD system structure by mediating interactions between the user and the system, the system and the data base, and the user and the data base. The executive system user interface supports comprehensive "HELP" capabilities, the HDL, and a logically integrated command language. Documentation describing the system and the CAD codes is maintained by the Design Executive in a hierarchical configuration. User access to this documentation structure is facilitated by the intelligent intervention of the Design Executive.

#### HDL-Design Specification

The structure and behavior of a digital system must be described in various ways at many levels to completely characterize a design. Existing "languages" such as ISP [9], DDL [10], HDL [11] and are usually associated with specific descriptions of architecture, system behavior, system structure, logical structure, circuit structure, logical behavior, or physical structure. These descriptions lack the generality required to support VLSI design, since most are applicable to one level of design and a particular design style and are not integrated into a complete design system.

Ultimately, a comprehensive HDL must be used to describe all levels of system behavior and structure; including normal and faulty electrical operation, logical and functional behavior, and physical structure. Organized in a hierarchical manner, the HDL should allow description of functional and physical entities and relationships among entities.

#### Synthesis and Functional Verification

Functional synthesis begins at the architecture level with a description of the overall behavior of a large system and interaction with the environment. As synthesis proceeds, more structural detail is added at system, logic, and circuit levels in the form of interconnection structures. Concurrently, behavioral specifications are developed at each level of the design hierarchy.

During synthesis, the actual behavior of the system must be forced to match the specifications. To effect the match, models are required for computational algorithms and interconnects, and procedures for mapping one into the other. Models must be expressible in a computer-readable form in order to manage complexity with an interactive computer support system, permit the separation of structure from associated behavior, and support direct fabrication of the synthesized system.

Simulation

Simulation, or dynamic design verification, is the process of calculating the behavior of a system within an environment specified by the designer. The objective of simulation is to verify that the system will perform correctly in an operational environment.

Ideally, simulation should be multilevel; i.e., consider larger results at a less detailed level with the capability of simultaneously simulating subcircuits more exactly. This concept

complements the hierarchical design implementation. In a hierarchical simulator, the basic elements are more complex than simple modules or gates. Models are formulated for entire IC's and include gate, function, and transistor models. Parts of a system are simulated in great detail while other parts are simulated abstractly. Models are written in high-level languages, with models for MSI or LSI blocks only slightly more complex than the model for a simple NAND gate. Thus a system of many thousands of devices or gates can be reduced to a few hundred hierarchical models, making it feasible to simulate entire VLSI systems.

#### Testability

Testing is the process of verifying that a system is operating properly. Because of the increased complexity of VLSI circuits, the difficulty of testing is substantially increased. To ensure that a circuit can be tested, design-for-testability techniques must be used throughout the design process.

Approaches to design-for-testability fall into two major categories: testable design styles and testability measure analysis. The use of a testable design style guarantees that generating tests for a circuit will not be impossible. Testability measure analysis computes the difficulty of controlling and observing each internal node from primary input or output pads. This information is used in the design process to locate potential circuit testing problems and provide feedback about the effect of circuit modifications on testability.

In developing a test sequence, knowing that a good test exists is not the same as knowing the test. Deriving a test is the task of the test sequence generation phase. Given a digital circuit and a set of possible faults, a series of input signals (vectors) is generated that will force any faulty circuit to behave differently from the fault-free circuit. The test sequences depend not only upon the intended function of the circuit but also upon the fault set assumed. Test-sequence generation must proceed concurrently with the hierarchical design process.

#### Physical Design Aids

Physical design aids include tools to partition, place, and interconnect circuit components and verification tools to ensure the synthesis has been performed properly. To provide an optimum system, physical design must be considered in parallel with functional design and testability.

**Partitioning**—Partitioning programs operate on both functional and physical entities. Partitioning techniques must function in both the initial design and detailed implementation phases. During top-down design, partitioning aids are needed for hierarchical decomposition. During bottom-up implementation, partitions are modified based on lower level implementations.

In addition to optimizing a circuit element assignment based on size and external pin connections, parameters such as circuit speed, power dissipation, and functional groupings must be considered. The partitioning aids must be capable of optimizing any of these quantities as a function of the others.

**Symbolic layout**—Provides a shorthand method for manually sketching a circuit layout using specified symbols to represent

DEF011956

various types of transistors and interconnections. The initial layout can be performed on grided paper or directly on an interactive graphics system. During layout, the designer is not concerned with the geometric design rules. Symbolic layouts are postprocessed using computer aids which expand and relocate all symbols and interconnects based on the geometric design rules for the specified technology to provide mask artwork files.

**Physical layout**—The physical layout phase of IC design involves positioning and interconnecting electrical components. In hierarchical design, the concept of a component is generalized. Components may range in complexity from primitive transistors and fundamental gates to microprocessors that can be combined to form a microcomputer.

Hierarchical layout is applicable to a wide range of physical design levels. At the highest design level, the shape of LSI components may be adjusted and combined to form a VLSI circuit. At the lower design levels, the same algorithms may be used to combine gates into registers or transistors into gates.

**Topological analysis**—Because of the high costs and long lead times involved in the fabrication of IC's, it is important to verify the correctness of a circuit design before manufacture. Verification tools must ensure correct physical mask layout, functional operation, and that electrical characteristics are within specified tolerances.

Topological analysis tools can be used to verify correctness of physical layout data by examining the artwork data and the circuit inputs and outputs. Design-rule checking codes perform geometrical, logical, and topological operations on artwork data and compare the results with design rules for the specified technology. Connectivity and electrical parameter data are used to reconstruct a detailed circuit including parasitic electrical components. This reconstructed circuit can be used in performing an accurate functional and timing analysis.

#### Circuit Design Documentation

After a circuit or system design has been completed, the design can be documented by collecting information from the data base generated during the design process. Design documentation includes information for fabricating the IC's and system as well as information for archiving the completed design to support later modifications. Computer aids should collect manufacturing and archival information and generate appropriate files, specifications, and reports.

#### Data Base

In general, the distinction between programs and data is that programs are active, data is passive (i.e., programs operate on data). The necessity for supporting the initial design, design changes, and providing software transportability dictates consistency in data handling procedures and mechanisms. Therefore, an efficient data base is vital for coordinating the various functions. As systems become more complex, it is imperative that an overall data base be established and maintained to ensure consistency in design and to eliminate duplication of information.

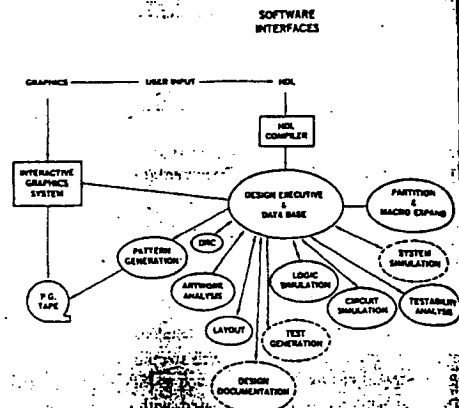


Fig. 8. Sandia hierarchical CAD system.

The data base must be designed in a flexible and modular fashion to provide upward compatibility with computer hardware and the evolving CAD software. Modularity is important where various portions of the data base may reside at more than one computer node. In addition to the actual data used by CAD programs, the data base should contain all necessary information to properly document elements being designed.

#### SANDIA CAD SYSTEM IMPLEMENTATION

A CAD System meeting the criteria outlined above is being implemented in phases at Sandia National Laboratories to provide design capability for LSI circuits initially, with evolutionary expansion and enhancement as required to support VLSI designs.

The design procedures consist of: (1) identifying critical aids in the design process based on user needs, (2) acquiring existing software when available and subsequently modifying it to meet system and user needs, (3) developing new aids with appropriate expansion capabilities to support hierarchical design, and (4) integration of all aids into the design system framework. A block diagram outlining the present system implementation is shown in Fig. 8. The solid blocks represent areas where design support is presently available, and the dashed blocks indicate work in progress. Brief descriptions of the hardware and software are given below.

#### Hardware

The software is supported on a dedicated DECSYSTEM 20 computer system containing 1 million words of main memory, 160 million words of disk files, and two 9-track 800/1600 bit/in, 75 in/s tape drives. Two Applicon 860 interactive graphics systems and a Versatec 36" electrostatic plotter are connected to the DECSYSTEM 20. A VAX 11/780 computer containing 4 megabytes of main memory, 323 megabytes of disk files, and 2 9-track 800/1600 bit/in, 125 in/s tape drive is interfaced in a network configuration to provide additional computing capability. Communication among the computer



UARY DANIEL AND GWYN: CAD SYSTEMS FOR IC DESIGN

and the Appicon systems is supported by DECnet. Both dial-up and direct-wired access to the system provide support for alphanumeric and graphics terminals.

#### Design Software

A Universal HDL (UHDL) is being developed in essentially three stages consisting of: (1) language formation by phrase concatenation, (2) redundancy removal, and (3) consolidation and rewrite of the compound language. During the first step, the SDL [11] language (a PASCAL-like description) is used to describe system structure. Other languages are appended in phrase form to describe functional behavior. After gaining experience with the composite languages and after the deficiencies and required enhancements have been identified, a language for the initial UHDL will be written. Continuous refinement and enhancement of the UHDL will be required to meet design and documentation needs.

Translation from UHDL descriptions (initially SDL) to the required formats for simulation, layout, and verification programs is performed by the Design Executive. The Design Executive also manages the data flow between the Data Base files (outlined below) and each of the design aids.

Design synthesis aids currently available include MINI [12] and DDA [13] (Digital Design Aids). MINI, a Programmable Logic Array (PLA) minimization program uses a branch-and-bound algorithm to produce the optimal two-level realization of a set of Boolean equations. DDA provides synthesis of a sequential-state machine based on flow diagram, state assignment, and flip-flop type.

Logic-level design consists of defining, in complete detail, all components (gates, cells, modules, etc.) and interconnections required to implement a specified function. The SALOGS [14] (Sandia LOGic Simulator) program performs true-value 4-state logic simulation (true, false, undefined, and high impedance) and 8-state timing simulation (four states plus transition to each of the four states) as well as states-applied and gate activity analysis and fault simulation. SALOGS is used for dynamic design verification, test sequence evaluation, and fault coverage calculations. Race and hazard analysis is available and the program is capable of using library or user-defined models.

A typical input/output for SALOGS is shown in Fig. 9 for a small logic circuit. The input consists of a connectivity description of logic gates or cells contained in the model library. The plotted output describes the logic signal changes as a function of time for each of the nodes indicated.

Although approximate timing simulation is accomplished in SALOGS through the use of extra states to represent logic values in transition, more exact timing simulation is possible with MOTIS-C [15] and SIMPIL [16] for MOS and  $I^2L$  circuits, respectively. These programs use tables to describe active devices in a circuit instead of models with complex equations. Both programs provide more accurate analysis than logic simulators and run more efficiently than circuit simulation codes such as SPICE.

A more exact circuit analysis is obtained by using the SANCA (Sandia Circuit Analysis) program circuit simulation program based on SPICE. This program simulates MOS and bipolar

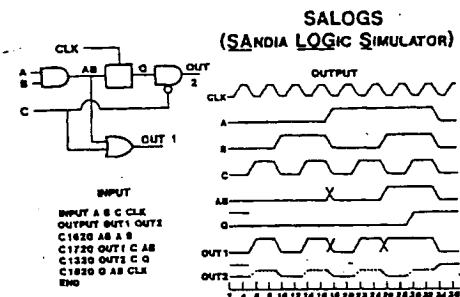


Fig. 9. Typical logic simulation input and output using SALOGS.

circuits, including analog and digital circuits, by numerically solving the network equations to calculate the desired voltages and currents. The cost of increased accuracy (over timing simulation) is a decrease in both execution speed and circuit size that can be analyzed. Model libraries [17] for circuit analysis are maintained to reflect varying levels of accuracy and different modeling philosophies. Circuit models based on device physics, empirical behavior, and hybrid approaches are all used in the circuit simulation environment. SANCA is an enhanced, quasi-interactive, topological analysis program containing model information for specific Sandia technologies and output plot routines.

Physical mask layout is performed using the SICLOPS [18], [19] (Sandia IC Layout Optimization System) and SLOOP [20] (Standard cell LayOut Optimization Program) programs to automatically place and interconnect generalized circuit elements. Two different forms of mask layout are used. A hierarchical mask layout system has been developed and used for specific designs. Hierarchical layout is applicable to a wide range of design levels. At one extreme, LSI components can be combined to form a VLSI circuit. At the other extreme, the same algorithms can be used to combine gates into registers based on a standard cell layout. This hierarchical approach relies on the use of SICLOPS to perform an automatic layout of an integrated circuit using rectangular-shaped blocks. The blocks must be rectangular and can be of arbitrary size and aspect ratio. The program automatically places the blocks and performs the interconnections. Each of the blocks can contain nested subblocks to any depth to provide a hierarchical layout. SICLOPS can also be used to layout a thick film hybrid circuit, as shown in Fig. 10.

The more conventional layout using standard cells placed in rows with interconnection between cell rows in routing channels is supported by SLOOP. In addition to designing modules used by SICLOPS, conventional standard cell IC layouts can be designed. The basic program can be used in an interactive mode to optimize the design, will perform a 100-percent completion of all interconnections by expanding the chip size as required to complete interconnections, contains hierarchical interfaces for use with the SICLOPS code, and supports multiport gates or cells. A typical IC design using SLOOP is shown in Fig. 11.

DEF011958



10 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. CAD-1, NO. 1, JANUARY 1982

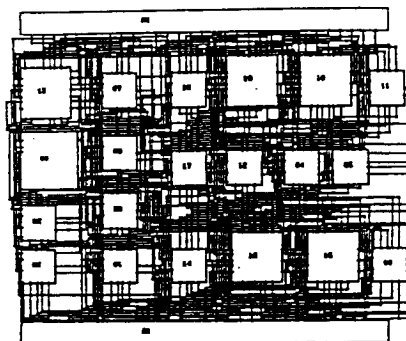


Fig. 10. Thick-film hybrid circuit layout obtained using SICLOPS.

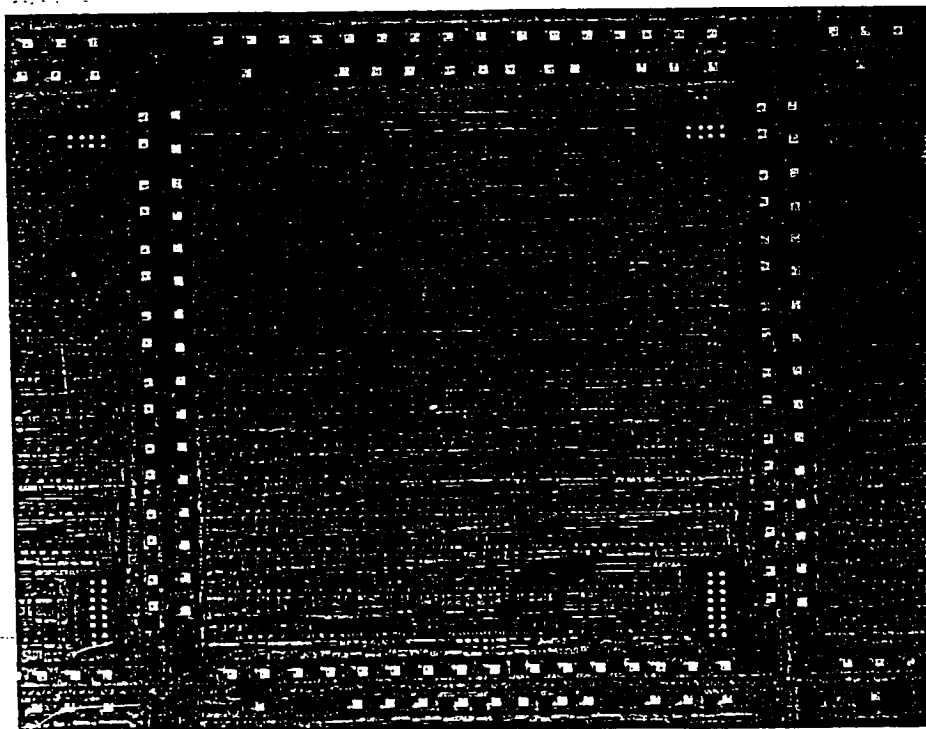


Fig. 11. Typical standard cell IC design using SLOOP.

DEF011959

JANUARY 1988 DANIEL AND GWYN: CAD SYSTEMS FOR IC DESIGN

The LOGMASC (LOGical MASK Checking) [21] program is a tool for design rule checking of IC masks. This program produces Boolean logic combinations of mask levels and is based on pattern recognition techniques. LOGMASC provides essentially all design rule checks which can be performed manually and provides pattern identification and extensive topological and geometrical information concerning the interrelationships of entities in the file. False errors are minimized by selectively applying design rule checks based on the circuit use of the particular patterns begin checked. Many of the algorithms can be used to solve other mask analysis problems in addition to the design rule checks.

During the circuit mask design process, transistor current or voltage gains, resistance, and capacitance values are scaled and converted to area definitions based on the nominal circuit parameter values used for circuit analysis. The actual sizes of the circuit elements and devices are adjusted to fit layout constraints based on minimum spacing, relative element partitioning, and size requirements. The actual circuit defined by the mask layout may be quite different than the original circuit schematic. For example, the interconnection distances between transistors introduce resistances and capacitances into the circuit which were not included in the initial circuit simulation. Additional parasitic transistors and diodes can be unintentionally introduced into the circuit which can cause improper operation. The CMAT [22] (Circuit MASK Translator) code operates on circuit mask information, using pattern recognition to recognize and transcribe the mask areas into the circuit elements. CMAT employs the basic Boolean operations algorithms contained in LOGMASC and performs the required scaling operations to obtain circuit element values.

SCOAP [23] (Sandia Controllability Observability Analysis Program) is used to evaluate the testability of a circuit after the basic circuit design has been completed. This analysis is performed prior to layout and is based on a simple analysis of the logic interconnectivity. The SCOAP program calculates the ease or difficulty of setting an internal node from a primary input to a zero or one and the difficulty of observing a logic value at any internal node from a primary output. The zero and one values for controllability may be different depending upon the exact circuitry and interconnection of the logic gates. For nodes having very high values for controllability or observability changes are usually required in the logic circuitry to reduce the controllability and observability numbers. This reduction is required, since the numbers are proportional to the testing difficulty. For nodes with high controllability/observability measures, additional input or output test points or signal multiplexing may be required to set logic gates or observe node logic values to achieve circuit testability. At present, a structured design for testability, such as the IBM LSSD [24] approach, is not required. However, since the designer is responsible for designing a testable circuit, techniques equivalent to the LSSD approach may be used in the design based on the testability analysis information.

The testability analysis program is used for both combinational and sequential circuits. Combinational measures basically relate to the number of different line assignments or input assignments which must be made to set an internal node to a

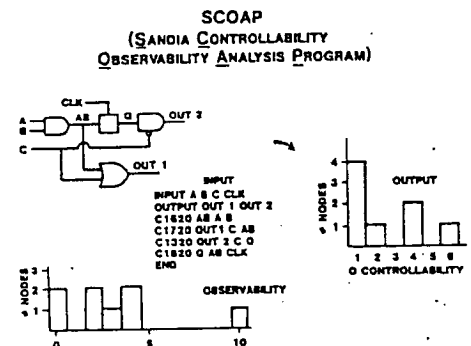


Fig. 12. Testability analysis using SCOAP.

given value. For sequential circuits, the measures relate to the number of clock cycles or time frames required to propagate a specified logic value from an input pad to a node or from a specific node to an output terminal.

An example at testability analysis is shown in Fig. 12. All input nodes are directly controllable as indicated for the four nodes with a "zero controllability" value of one. The most difficult node to control is usually an output node; for this example, output 2 has a zero controllability value of six. The observability is shown in the lower left of the figure. Again, the most observable nodes are the output nodes as indicated by the two nodes with an observability value of one. Ordinarily the most difficult nodes to observe are the input nodes and, therefore, the nodes A, B, C, and CLK should be the least observable. In this case, the CLK signal is input to a transmission gate between nodes AB and Q; a change of state at node AB must occur to observe the clock signal and, therefore, the CLK signal has the large value of 10 for observability.

#### Data Base

Because of the initial need for integrating loosely related CAD software and evolving design system specifications, a data-file base has been developed. The Sandia CAD data structure [25] is in the form of a "deciduous tree"; that is, one that can lose its leaves. This structure is a two-dimensional network with several restrictions. In one dimension the data structure forms a shallow tree one level deep. That is, any number of data types can be associated with a given node, but they may not be broken down further. In the other dimension, the structure is a network of nodes and subnodes to any depth. The leaves on the tree are the equivalent of small sequential files of data which can be retrieved by the user application program. This data file approach forces the designer to conform to a hierarchical structure while maintaining the freedom to store test files, documentation, etc., in the same file.

#### SUMMARY AND FUTURE PLANS

Although a basic computer-aided LSI design capability has been established and is used by design engineers, the system is continuously evolving as new capabilities and enhancements

DEF011960

are added. The modular programs and data structures, as well as the flexibility designed into the overall system framework, tend to minimize the cost for system modification as requirements change. In addition to continuous enhancement of existing programs, new aids for design synthesis, hierarchical simulation, symbolic layout, and test sequence generation will be developed. A continuing emphasis is placed on integrating the CAD software into a complete design system.

## REFERENCES

- [1] G. Kron, *Tensor Analysis of Network*. New York: Wiley, 1939.
- [2] —, "A set of principles to interconnect the solutions of physical systems," *J. Appl. Phys.*, vol. 24, pp. 965-980.
- [3] A. F. Malmberg, F. L. Cornwell, and F. N. Hofer, "NET-1 network analysis program," Los Alamos Rep. LA-3119, 1964.
- [4] R. W. Jensen and M. D. Lieberman, *The IBM Circuit Analysis Program*. Englewood Cliffs, NJ: Prentice-Hall, 1968.
- [5] A. Feller and M. D. Agostino, "Computer-aided mask artwork generation for IC arrays," in *Proc. 1968 IEEE Computer Group Conf.*, pp. 23-26.
- [6] A. Feller, "Automatic layout of low-cost quick-turnaround random-logic custom LSI devices," in *Proc. 13th Design Automation Conf.*, pp. 79-85, June 1976.
- [7] G. Penky, D. N. Deutch, and D. G. Schweikert, "LTX-A system for the directed automatic design of LSI circuits," in *Proc. 13th Design Automation Conf.*, pp. 399-407, June 1976.
- [8] E. Cohen, Program references for SPICE 2, Memo ERL-M592, Electronics Research Laboratory, Univ. California at Berkeley, June 1976.
- [9] M. R. Barbacci et al., The ISPS computer description language, Tech. Rep. Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, PA, Mar. 1978.
- [10] W. E. Cory et al., An introduction to the DDL-P language, Tech. Rep. 163, Computer Systems Lab., Stanford Univ., Stanford, CA, Mar. 1979.
- [11] W. M. van Cleemput, A structural design language for computer-aided design of digital systems, Tech. Rep. 136, Digital Systems Lab., Stanford Univ., Stanford, CA, Apr. 1977.
- [12] J. Abraham, MINI-A logic minimization program, Univ. Illinois, Urbana, IL.
- [13] DDA-Digital design aids for state machine synthesis, Tektronix, private communication.
- [14] J. M. Acken and J. D. Stauffer, "Logic circuit simulation," *IEEE Circuits Syst. Mag.*, vol. 1, no. 2, pp. 3-12, June 1979.
- [15] S. P. Fan et al., "MOTIS-C: A new circuit simulator for MOS LSI circuits," in *Proc. IEEE 1977 Int. Symp. on Circuits and Systems*, pp. 700-703, Apr. 1977.
- [16] G. R. Boyle, *Simulation of Integrated Injection Logic*, Memo. UCB/ERL M78/13, Electron. Res. Lab., Univ. California at Berkeley, Mar. 1978.
- [17] D. R. Alexander, R. J. Antinone, and G. W. Brown, *SPICE 2 Modeling Handbook*, BDM/A-77-071-TR, May 1977.
- [18] B. T. Preas and W. M. van Cleemput, "Placement algorithms for arbitrarily shaped blocks," in *Proc. 16th Design Automation Conf.*, June 1979.
- [19] B. T. Preas and W. M. van Cleemput, "Routing algorithms for hierarchical IC layout," in *Proc. Int. Symp. Circuits and Systems*, July 1979.
- [20] SLOOP (Standard-Cell Layout Optimization Program), Sandia Lab., to be published.
- [21] B. W. Lindsay and B. T. Preas, "Design rule checking and analysis of IC mask designs," in *Proc. 13th Design Automation Conf.*, June 1976.
- [22] B. T. Preas et al., "Automatic circuit analysis based on mask information," in *Proc. 13th Design Automation Conf.*, June 1976.
- [23] L. H. Goldstein, "Controllability observability analysis of digital circuits," *IEEE Trans. Circuits Syst.*, vol. CAS-26, Sept. 1979.
- [24] E. B. Eichelberger and T. W. Williams, "A logic design structure for LSI testability," in *Proc. 14th Design Automation Conf.*, pp. 462-468, June 1977.
- [25] J. D. Stauffer, "SADIST (The Sandia Data Index Structure)," Sandia National Lab., Albuquerque, NM, SAND80-1999, Nov. 1980.



Marvin E. Daniel was born in Dexter, KS, on December 19, 1938. He received the B.S.E.E. degree from Kansas State University in 1961, the M.S.E.E. degree from University of New Mexico, Albuquerque, in 1963, and the Ph.D. degree in electrical engineering from Oklahoma State University, in 1966.

In 1961, he joined Sandia National Laboratories, Albuquerque, where he has worked in a variety of positions including: design of test equipment, application of minicomputers to automated test systems, development of mathematical models for CAD programs, modeling of radiation effects in semiconductor devices, analysis of weapon systems, economic studies of fusion and fossil fuel energy systems, and for the past two-and-one-half years has been supervisor of the Computer-Aided Design Division in the IC Design and Fabrication organization. He is the author of numerous papers and articles, particularly in the areas of programs and models for computer-aided design.



Charles W. Gwyn (M'68-SM'75) received the B.S. degree in electrical engineering from the University of Kansas, Lawrence, in 1961, and the M.S. and Ph.D. degrees in electrical engineering from the University of New Mexico, Albuquerque, in 1963 and 1968, respectively.

He joined Sandia National Laboratories, Albuquerque, in 1961, as a staff member where he was engaged in many phases of nuclear radiation effects studies, including the study of radiation effects on semiconductor devices and analysis of the vulnerability of electrical systems to a radiation environment. In 1973 he was assigned supervisory responsibility for the Computer-Aided Design and Analysis Division, and for developing computer techniques for use in the design, analysis, and layout of the large scale integration (LSI) circuits. Currently he is department manager for the IC Design Department and has responsibility for method oxide semiconductor (MOS) and bipolar IC design and testing, development of computer aids, and development and procurement of special semiconductor devices.

DEF011961

## The CMU Design Automation System

## An Example of Automated Data Path Design

A. Parker, D. Thomas, D. Siewiorek, M. Barbaresi, L. Hafer, G. Leive, J. Kim

Carnegie-Mellon University  
Departments of Electrical Engineering and Computer Science  
Pittsburgh, Pennsylvania 15213

## Abstract

This paper illustrates the methodology of the CMU Design Automation System by presenting an automated design of the PDP-8/C data paths from a functional description. This automated design (using synthesis techniques) is compared both to DEC's implementation and the Intersil single chip implementation.

## 1. Introduction

As it is becoming possible to integrate larger numbers of logic components on a single chip, the need for more powerful design aids is becoming apparent. Indeed, these aids must be capable of supporting a designer from the system level of design down to the mask level. In this way the systems level designer can become more aware of the implications of higher-level design trade-offs on implementation properties such as silicon area, power consumption, testability, and speed, and be able to make more timely use of new technologies. The ultimate goal of the Carnegie-Mellon University Design Automation (CMU-DA) System [12] is to provide a technology-relative, structured-design aid to help the hardware designer explore a larger number of possible design implementations. Inputs to the system are a behavioral description of the system to be designed, an objective function which specifies the user's optimization criteria, and a data base specifying the hardware components available to the design system.

The CMU-DA system differs from other design automation systems because the input design description is a functional specification. Such a specification provides a model that, while accurately characterizing the input-output behavior desired for the implementation, does not necessarily specify its internal structure. The system software collectively performs the synthesis function by transforming the input functional description into a structural description. The design process involves binding implementation decisions in a top-down manner as a design proceeds through the design system. More structural decisions are made at each level until a complete hardware specification is obtained, with the most influential design trade-offs being performed first in order to cut down the design search space.

The purpose of this paper is to illustrate the methodology of the CMU-DA system. The results given here are worst case - many optimizations, which are straightforward have not been implemented yet; research is in progress on others. The design of the data path of a DEC PDP-8/E [5] from the ISP level through to a TTL and standard cell design will be discussed. Only the subset of the full DA system which is presently implemented has been used for this example. The

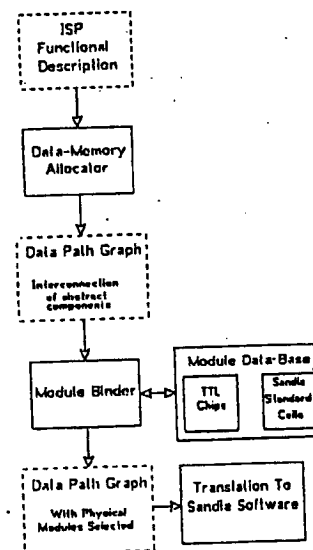


Figure 1: The CMU Design System

components are shown in Figure 1.

The PDP-8/E is first functionally described using the ISP language [1]. A data-memory allocator [7] is used to generate the structure of the data paths from the functional description. This allocation is in terms of abstract logic components. The next step in the design is to bind physical modules to the abstract design from a module database [9]. This step provides the system with the capability of designing relative to new technologies. This binding will be illustrated both in terms of TTL chips and the CMOS standard cells in the Sandia design system [10]. The standard cell output of the CMU-DA system is then translated for input to the Sandia design system. At this point, chip area can be calculated and detailed timing information can be gathered using SALOGS [3].

The paper will conclude by comparing these two alternative designs to commercial implementations.

This research is supported in part by NSF Grant MCS77-09730 and Army Research Office Grants DAAG29-76-G-0224 and DAAG29-78-G-0070.

## 2. The PDP-8/E Example

As the CMU-DA system has evolved, more complex design examples have been used to observe its performance. The use of the POP-R/E is a major step in two ways: 1) It represents about a 10-fold increase in the circuit complexity over previously reported example designs [8] and 2) It is a commercially available system that has been implemented in various technologies by different manufacturers over several years. Thus, it is possible to compare a non-trivial automated design to designs done by several designers with different logic components.

The PDP-8/E is one model in a family of computers having nearly identical ISP descriptions. (That is, they implement nearly the same instruction set, with different hardware structures). A portion of the ISP description used by the automated design system is shown in Figure 2.

The description begins by declaring the memory and processor state. A 13 bit link-accumulator (Lac) register is defined but can alternately be accessed as the one bit link (L) and 12 bit accumulator (Ac). Next, instruction interpretation is defined using the declared memory and registers. After the instruction fetch and increment of the program counter in the instruction interpretation section, instruction execution (excc()) is called. Illustrated here are the six memory reference instructions. Not shown but implemented in the automated designs are the effective address calculation, input/output instructions, and the operate microinstructions. Note that the Mb=Hp[Pc] instruction fetches the location pointed to by the Pc and places it in the Mb register. No mention need be made of transfer of the current Pc to a memory address register, which is a part of the hardware structure.

The ISP description [1] is compiled into a representation which is machine readable by the data-memory allocator. The next sections will discuss the data-memory allocation (where an abstract data path is synthesized), a module binder that illustrates how the CMU-DA system can design relative to new technologies, and a translator to Sandia's SALOGS simulator [10].

### 3. The Allocation Process

The data-memory allocators perform a mapping function from the algorithmic (ISP) description to the data-path part of the hardware implementation, which is called a data-path graph. The data part consists of the data-storage elements, data operators, and data paths necessary to implement the operations specified in the algorithmic description. Due to the characteristics of the ISP language this mapping may be multi-valued in either direction, rather than a simple one-to-one translation.

The POP-8/E design described here was produced by a data-memory allocator which uses the distributed-logic design style. This style of (for approach) to design encompasses design with small and medium scale integration components. As pointed out earlier, the allocator itself is technology relative and the mapping onto specific integrated circuit packages is performed by a separate module binder program. The process referred to as allocation throughout the remainder of this paper is a synthesis of logic using generic logic elements: data paths, operators, registers, and multiplexers, all of any bit width.

The procedure used by the allocator might be compared to a two-pass compilation. The first pass may be considered a syntax or feasibility check. The allocator inputs a parsed ISP description, constructs data structures analogous in function to symbol tables, and enforces constraints necessary to insure that the data-storage locations, logical mappings, and

```

pop8 is BEGIN
| The basic POP-B instruction set (without extended arithmetic
| element) is implemented. No I/O devices are included in the
| description.

| NP State
Mp (#0:#7777):#011>, | Main memory
Mn-011>, | Memory buffer

| PC State
lac-012>, | Line as register
Lcs to lac-0>, | Link bit
Ac-011> to lac-112>, | accumulator
Pc-011>, | Program counter
lac.pc-011>,

| Instruction Interpretation
start :=
BEGIN
gc = 1 NEXT
rim()
END,

run\Instruction.Interpretation :=
BEGIN
IF gc =>
gc :=
Run(
Mp(Pcl) to Lpc = Pc NEXT
Pc = Pc + 1 NEXT
exec() NEXT
IF interrupt.enable AND interrupt.request =>
BEGIN
MpDI = Pc NEXT
Pc = 1
END NEXT
RESTORE rim
LNO
END,

| Instruction Execution
exec(instruction,operation) :=
BEGIN
If = Mv-012> NEXT
IF (Ir GED #3) AND (Ir LEQ #5) == null NEXT
IF Ir LEQ #2 == Mn = Mp(mal) NEXT
DECODE Ir ==
BEGIN
#0 to add:= Ac = Rc AND Mn, | Add
#1 to lsd := lsr = lsc = Mn, | load (TC)
#2 to lsr := RGIN | swap if zero
Mn = Mn + 1 NEXT
IF Mn EOL & == Pc = Pc + 1
END,
#3 to dca := BEGIN | Deposit and
Mn = Rc NEXT | clear Rc
Rc = 0
END,
#4 to jmp := BEGIN | Jump to
Mn = Pc NEXT | subroutine
Pc = mn + 1
END,
#5 to jmp to Pc = Mn, | Jump
#6 to call | I/O execution
#7 to opst | Operate
microinst.

END NEXT
IF (Ir GED #3) AND (Ir LEQ #4) == Mp(mal) = Mn
END
I End of description

```

Figure 2: ISP Description of the PDP-8/E



Input/output interface characteristics specified in the description can be implemented in hardware. If no errors are encountered, it proceeds to allocate the basic data-storage structures called for in the description, and any additional data paths, storage, and operators necessary to implement variable-accessing schemes described by ISP. The second pass may be considered as the semantic phase, with the activity of code generation replaced by the allocation of data paths, operators, and additional storage as needed to implement the actions described in the ISP description. Parallelism analysis is performed at several levels to warn the user of error conditions and determine constraints relating to optimization of hardware. The allocation is then completed by the addition of multiplexing where required.

However, allocation differs from compilation in that in a compilation one is concerned with implementing the specified data operations on a fixed data part whose capabilities are known *a priori*. In allocation, the allocator must be able to recall and utilize the capabilities of a data part which is being dynamically created. The allocator thus works from the inside out, first creating the data storage and access structures, and then adding the necessary data paths and operators to perform the described data operations. Finally, the output of the allocator is a non-planar directed graph, rather than a linear list of compiled instructions.

The first version of the allocator is experimental, and it performs only minor optimizations on the allocated hardware. It has been designed to investigate the feasibility of performing the mapping from ISP to hardware, the types of data structures needed for allocation, and areas where optimizations are possible in future, more sophisticated allocators.

The allocator has been designed as a possible skeletal structure for future allocators in order to standardize input/output formats and data structures.

#### 4. Performance of the Data-Memory Allocator

The allocator program was run using the ISP description of the PDP-8/E, and the resultant data paths are shown in Figure 3. A binding of modules was done by hand to compare the results of the allocator to the original DEC design (Figure 4) [5].

It is difficult to compare the automated PDP-8/E data-path design with the original DEC design for three reasons. First, the ISP description input to the allocator declares as registers some values the PDP-8/E uses but never stores explicitly in registers, such as the effective address. These show up as registers in the allocator's design. Second, the allocator designs distributed logic, and the DEC design was done in the central-accumulator design style. (That is, this allocator does not contain the design rules for large scale collapsing of the data paths into a central-accumulator style of design [13]). Third, the DEC design has assumed a boundary between the control and data-memory parts of the design, but the boundary is different from that imposed on the allocator by the ISP description. Thus, some tests, flags, and registers which must be declared explicitly in the ISP description are part of the control in the DEC design.

The main reason for the difference in design seen from the block diagram level of Figures 3 and 4 is that the design styles are different. The multiplexing is used in different ways. In the DEC version, the operators are shared, and are used to provide no-op paths from one register to another. In the CMU version, only registers are shared and use multiplexed inputs. The ISP language is partially the source of this disparity. In ISP, the user can repeatedly use register A as a destination from various sources. However, the expressions A+B and C+D do not imply nor discount a single adder. Other differences in the design include the use of multiplexers for shifting in the

DEC design, and use of true/complement 0/1 chips for creating complements. "ORing" of the MQ and AC registers in the DEC version is done within the multiplexing hardware. Constants are often created in one place and gated over already existent data paths to the registers. In the CMU version, these constants are multiplexed at the register inputs.

In spite of these differences, estimates of chip count indicate that the allocator produces a path graph which would require 392 more integrated circuit chips than the DEC designers used for the data paths and registers. These estimates were done by hand to gauge the performance of the allocator; an automatic binding is discussed in the next section. These estimates were made using the same 1970 technology chip set the DEC designers had to deal with. The 392 excess hardware can be found in multiplexers which connect the registers, the extra registers declared in the ISP description, and duplicated operators like increment and add. Much of this excess can be attributed to the lack of optimization capability in the allocator algorithm. Future, more sophisticated allocation algorithms, coupled with the capability for high level optimization [12] available in the complete CMU-DA system will be able to significantly improve the data part design.

Further analysis of this design is in progress and includes a manual implementation of the control part. Comparisons of the DEC and CMU data-path speeds will then be possible.

#### 5. Module Binding

The module binding phase of the CMU-DA system employs the Module Data Base System (MDBS) and follows the data-memory allocation step. It has the task of translating the abstract links, memories, registers, and operations in a data-path graph into a design using physically realizable modules. A second pass of module binding will occur after control allocation.

At present the module binding portion of the design system is primarily a research tool that will be used to investigate automated module binding. A goal of this research is to model the module binding problem sufficiently to generalize this part of the CMU-DA system to handle a wide range of module types from LSI chips through Standard Cells. The implemented portions of MDBS were used to assist a designer in binding data-part modules in the CMU PDP-8/E data-paths produced by the Data-Memory Allocator described in Section 4. The results of the data-part module binding are compared to the DEC PDP-8/E design in Section 7. A similar comparison will be made to the standard-cell binding after those results are presented in Section 8.

#### 6. Organization of MDBS

MDBS consists of four sections shown in Figure 5: an I/O section that is responsible for translating between the internal and external forms of the path graph; a Module Data Base access mechanism; a command language interface; and the module binding mechanism.

The input/output section is the interface to other parts of the CMU-DA system. The path graph, generated by an allocator is placed in internal form for processing. The output file has the same format as the input path graph (with module binding information appended) and can be reread by the input section for additional processing.

The Module Data Base is a hierarchical data base that is distributed in various ASCII files. The highest level of the data base is the index which is read automatically during system initialization. The index contains pointers to all defined design-style sets, each of which is a collection of module sets appropriate for a given design style. A design style set file contains pointers to the actual module set information (the "Data Book") and summary information (typical cost, speed,

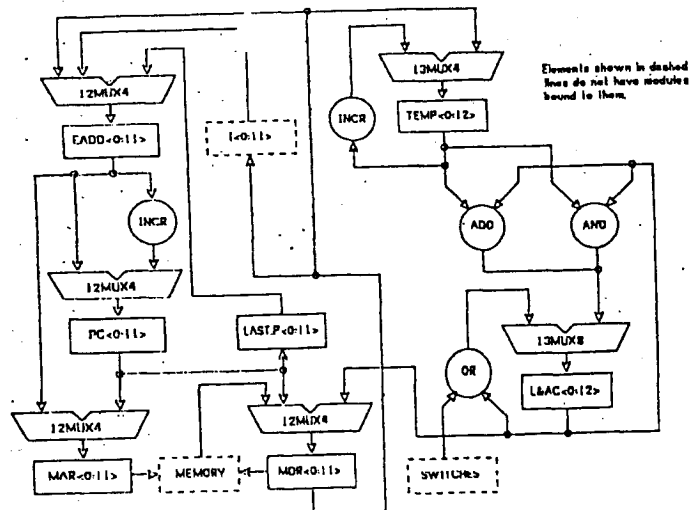


Figure 3: Allocator Generated POP-8/E Data Paths

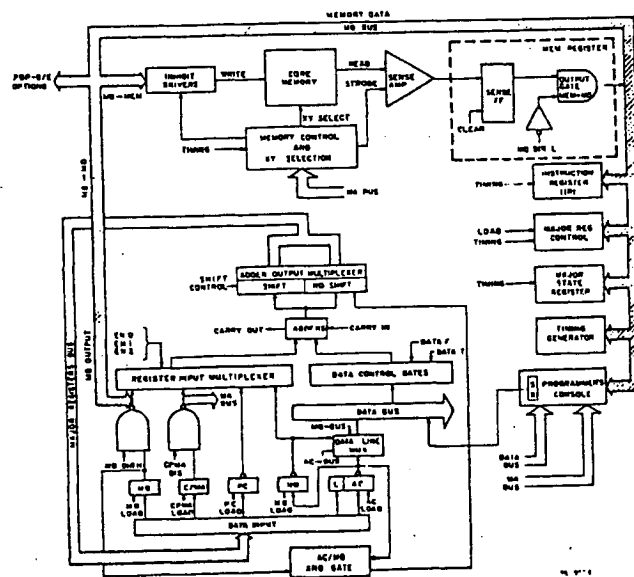


Figure 4: IXC PDP-8/E Data Paths

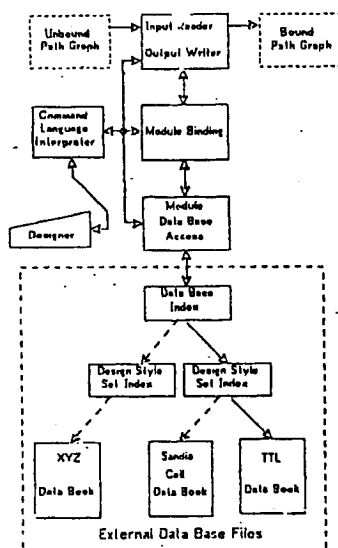


Figure 5: Organization of the Module Data Base System

load, and drive capabilities of modules in the set). Access to data base information during module binding occurs frequently and utilizes various levels of detail from basic type matching through specific delay and timing characteristics.

The Command Language Interpreter (CLI) is the experimenter/designer's interface to the module binding system. The CLI may be used only to select a file to be processed and direct the disposition of the resulting bound path-graph. The CLI also provides a number of tools to inspect the path-graph, modify the graph structure and bind designer selected modules to specific nodes of the graph.

The module binder applies transformations to both the path graph and module functions in order to match the desired behavior with the available building blocks. The graph transformations are localized decompositions or combinations of nodes that preserve the specific behavior. Module transformations are primarily combinations of nodes since modules cannot be physically decomposed. However, multifunction modules such as shift registers and ALUs may require partitioning of the non-conflicting functions of the same module into separate nodes of the graph.

The most prevalent type of graph transform is localized to a single node or several connected nodes of a similar type. Registers are usually decomposed into nodes of smaller bit width. Logical operators are usually modified by reduction to a canonical form, then synthesized with available module operations. Multiplexers and demultiplexers are frequently transformed from a single level to a multilevel form. Arithmetic operators (particularly the signed and complement arithmetic modes) require algorithmic decomposition.

A more complex type of transformation involves the combination of nodes of different types into single nodes capable of multiple functions. Shift operators and special purpose arithmetic operators (increment, decrement, and clear)

are generally combined with register functions in available module sets. These transformations often provide significant reductions in the graph complexity by elimination of constants and reduction of multiplexer size.

Operator node transformations primarily involve the application of axioms and identities to combine available modules into an aggregate that performs a desired function. Boolean identities and DeMorgan's Theorem will direct logical synthesis. Arithmetic mode transformations (for unsigned, signed-magnitude, two's complement, and one's complement) will be utilized to synthesize required modes from available modes.

For cases where single transformations on either the graph or the modules do not provide the desired match, an iterative approach will be utilized. The graph and the module transforms will be alternately applied until one or more matches are found, a cycle is detected in the transformations, no further gain is detected by applying transforms, or one of the system constraints (speed, cost, etc.) is violated by the resulting implementation.

A central goal of the CMJ-DA system is to produce designs that have been optimized toward the designer's objectives and fall within the external constraints. Module binding is the first operation in the design system that attaches actual costs to the implementation, and has specific speed, delay, and power information available. Therefore, evaluation of the bound design must be performed to insure compliance with the constraints. Critical constraints (i.e. constraints which the designs must meet) will be dynamically estimated by projecting the final value based on an extrapolation from the number of nodes remaining to be bound and the accumulated value for the nodes already bound; a true evaluation of the fully bound design will be made as a final pass to insure compliance with the constraints. The dynamic evaluation will be used to select between functionally identical module choices with different performance parameters.

## 7. PDP8 Baseline Example

A partial module binding of the PDP-8/E was done with the available pieces of MIXIS, which contain limited transform capabilities.

The module binding version contains a set of primitive operations that can modify the path graph structure (but not its behavior). The operations allow register nodes to be split at bit boundaries, operator nodes to be joined into single nodes with wider bit widths, nodes to be inserted, and nodes to be deleted. These operations allow the path graph nodes to be transformed to conform with the structure of available modules. As the MIXIS becomes more fully implemented, these primitive operations will be used to build larger scale path graph transforms that can be applied automatically.

The existing system aids the designer in producing correctly bound path graphs by the strict enforcement of rules concerning application of the structural modification primitives. These operations are restricted to minimize the possibility of modifying the behavior of the path graph. Some examples of these rules are:

- Nodes may not be deleted while connected to more than one link.
- Links may not be deleted while joining two nodes.
- Half links (i.e., signals to external sources) may not be deleted.
- Operator nodes may be joined only if they are of the same type.

A different but equally important kind of assistance is provided to allow the designer to display any aspect of the binding process. For example, the designer may display one module in the data base, all modules providing a specific function, or the entire data base.

The module choices were made by the designer using information from the path graph and a small TTL module data base from the distributed Design Style Set. The purpose of binding the data part with the existing system is to contrast the module selection with a hand designed PDP-8/E. This example provides a worst-case from which to judge the performance of MOBS as more capabilities are added.

Figure 6 was generated by the MOBS. The registers (named variables), operators, and multiplexors are identified in the "Comp." (Component) column. The "Device" column lists the name of the selected package. "Meds." lists the number of modules required to implement the component function. The term "module" refers to a separate functional unit in this context. There may be several modules contained in one package. "Pkgs." lists the number of packages required for each function. "Gates" is an estimate of the equivalent number of logic gates to implement the function. The percentage of the total gates required is listed in the "% Total Gates" column. The cost of each function implementation is computed as the basic cost for the number of packages required plus an overhead mounting cost of \$3.00 per package. The percentage of the total cost attributed to each function is listed in the "% Total Cost" column.

Comp.	Device	Meds.	Pkgs.	Gates	% Total Gates	Cost	% Total Cost
ENDD	SN74174	2	2	78	3.38	7.78	2.58
LNC	SN74194	3	3	185	8.58	11.87	3.87
LRST.P	SN74174	2	2	78	3.38	7.78	2.58
PC	SN74161	3	3	284	18.89	11.87	3.87
PCR	SN74174	2	2	78	3.38	7.78	2.58
PAR	SN74174	2	2	78	3.38	7.78	2.58
TEMP-8	SN7474	1	1	6	0.31	3.38	1.11
TEMP	SN74174	2	2	78	3.38	7.78	2.58
EDL	SN7485	3	3	93	4.87	11.37	3.77
EDL	SN7485	3	3	93	4.87	11.37	3.77
IMCR	SN7483	3	3	188	8.68	18.77	3.57
IMCR	SN7483	3	3	188	8.68	18.77	3.57
AND	SN7488	12	3	12	0.63	9.64	3.18
OR	SN7432	12	3	12	0.63	9.75	3.23
RON	SN7483	4	4	144	7.54	14.38	4.78
12MUX8	SN74151	13	13	156	8.17	46.87	15.48
12MUX4	SN74153	12	6	96	5.83	21.54	7.14
12MUX4	SN74153	12	6	96	5.83	21.54	7.14
12MUX4	SN74153	12	6	96	5.83	21.54	7.14
12MUX4	SN74153	12	6	96	5.83	21.54	7.14
13MUX4	SN74153	13	7	184	8.45	25.13	8.33
Totals		131	63	1989	100.00	301.54	100.00
Component Class				% Gates		% Cost	
Registers				38.48		19.17	
Operators				25.88		23.49	
Multiplexors				33.74		52.37	

Figure 6: Module Utilization For PDP-8/E Data Part

The choice of registers differed from the hand module binding in a few locations. The hand bound PDP-8/E used SN74194s (four bit universal shift registers) exclusively while the MOBS chose SN74174s (six bit D registers) when there was no requirement for the added shift capability. The Program Counter (PC) register was selected as three SN74161s (universal counters) based on the INC flag associated with the path graph node. The Accumulator (LAC) was first split into a one bit node and a twelve bit node, then the twelve bit part was allocated as three SN74194s. This is the same allocation that was done by hand. However, the

choice only callouts the shift ration flags (LSHFT and RSHFT). The increment requirement (INC) has effectively been partitioned out and must be bound separately.

The package count was 307 higher for the MOBS selection than for the DEC implementation (64 packages for DEC vs. 83 packages for MOBS). This agrees closely with the results obtained by selecting modules strictly by hand (refer to Section 4). The 307 difference is attributable to the different design styles used and the allocator's implementation of the design. Comparing the total cost of modules for the DEC implementation and the MOBS implementation (Figure 6), it is found that the costs also are 307 higher for the automated implementation, while the number of equivalent gates is 652 higher for the automated implementation. This indicates that MOBS chose modules with a higher level of integration than DEC did.

A comparison of the percentage of equivalent gates and the percentage of cost accumulated in three functional classes (registers, operators, and multiplexors) indicates surprisingly uniform comparisons. The percentage of both gates and cost is higher for registers in the MOBS implementation than in the DEC implementation. This trend is expected since the DEC PDP-8/E uses a central accumulator design style. Also, the slightly lower percentages for gates and costs in the operator class is reasonable for the DEC implementation. The most surprising comparison is the near identical percentage of gates devoted to data path routing (i.e. multiplexors) in the two designs implemented in different design styles. It would be expected that the central accumulator style would utilize more data path routing than the distributed style. This apparent anomaly is a clue to the area where the module binding can make local improvements in a path graph for distributed designs. By utilizing functions intrinsic to certain modules (such as the CLEAR on registers), constants and their associated data paths can be eliminated and improve the cost of implementing a design.

The TTL module binding using MOBS compares favorably with the DEC implementation and previous hand module bindings of the automated path graph. It is expected that much improvement in the package count (and the cost) is forthcoming as transforms and evaluation techniques are implemented in MOBS. However, TTL module binding is just one objective of a generalized design system. The following section discusses an approach to binding CMOS standard cells to a design with the objective of being able to automate and produce LSI designs.

## 8. Standard Cell Generation

The Sandia standard cell library [11] can also be used as physical modules to implement the automated PDP-8/E data part design. Then, using the Sandia software package [4, 10], it is possible to produce a simulation, insert faults, and perform automated cell placement and IC mask generation for a CMOS LSI chip implementation.

The standard cell binder used for this experiment was a small, automatic package which accessed a local data base of Sandia cells. This package only performed the essential transformations on the graph-expansion of nodes to match the standard cells. However, this package also gives us a worst-case measure for module binding performance.

The translation of the design from one environment, the module binder output, to another, the simulator input, involves both the expansion of multi-bit paths produced by the module binder to the single bit convention format of the simulator input and also the explicit identification of fan-out points. In addition to this latter process, termed resolving, the translator must generate gate specific parameters, such as propagation delays, by compiling capacitances to obtain a realistic simulation using SALOGS. Delay parameters are inserted in the





4. Case, G.R. and J.D. Stauffer. SALSIM - A Language for Control of Digital Logic Simulation. Proceedings of the 11th Annual Asilomar Conference on Circuits, Systems, and Computers, IEEE Circuits and Systems Soc., IEEE Control Systems Soc., Naval Postgraduate School, Univ. of Santa Clara, November, 1977, pp. 370-373.
5. DEC Staff. PDP-8/E Maintenance Manual. Digital Equipment Corporation, 1972. DEC-8E-HR1B-0
6. Electronics Magazine Staff. CMOS, Moving Along. *Electronics* 48, 10 (May 1975).
7. Hafer, L. Data-Memory Allocation in the Distributed Logic Design Style. Master Th., Carnegie-Mellon University, December 1977.
8. Hafer, L.J. and A.C. Parker. Register-Transfer Level Automatic Digital Design: The Allocation Process. Proceedings of the 15th Design Automation Conference, IEEE, 1978.
9. Leive, G.W. The Binding of Modules to Abstract Digital Hardware Descriptions. PhD Thesis Proposal, Carnegie-Mellon University, 1977.
10. Press, B.T. and C.W. Gwyn. Lecture For Contemporary Computer Aids to Generate IC Mask Layouts. Proceedings of the 11th Annual Asilomar Conference on Circuits, Systems, and Computers, IEEE Circuits and Systems Soc., IEEE Control Systems Soc., Naval Postgraduate School, Univ. of Santa Clara, November, 1977, pp. 309-317.
11. Sandia Staff. *Standard Cell User's Guide*. Sandia Laboratories, 1978.
12. Snow, E.A., D.P. Siewiorek and D.E. Thomas. A Technology-Relative Computer-Aided Design System: Abstract Representations, Transformations and Design Tradeoffs. Proceedings of the 15th Design Automation Conference, IEEE, June, 1978.
13. Thomas, D.E. and D.P. Siewiorek. Measuring Designer Performance to Verify Design Automated Systems. Proceedings of the 14th Design Automation Conference, IEEE, 1977, pp. 411-418.

# **ARTIFICIAL INTELLIGENCE TERMINOLOGY**

**a reference guide**

Colin Beardon

DEF079212

First published in 1989

**ELLIS HORWOOD LIMITED**

Market Cross House, Cooper Street,  
Chichester, West Sussex, PO19 1EB, England

*The publisher's colophon is reproduced from James Gillison's drawing of the ancient Market Cross, Chichester.*

**Distributors:**

*Australia and New Zealand:*

JACARANDA WILEY LIMITED

GPO Box 859, Brisbane, Queensland 4001, Australia

*Canada:*

JOHN WILEY & SONS CANADA LIMITED

22 Worcester Road, Rexdale, Ontario, Canada

*Europe and Africa:*

JOHN WILEY & SONS LIMITED

Baffins Lane, Chichester, West Sussex, England

*North and South America and the rest of the world:*

Halsted Press: a division of

JOHN WILEY & SONS

605 Third Avenue, New York, NY 10158, USA

*South-East Asia*

JOHN WILEY & SONS (SEA) PTE LIMITED

37 Jalan Pemimpin # 05-04

Block B, Union Industrial Building, Singapore 2057

*Indian Subcontinent*

WILEY EASTERN LIMITED

4835/24 Ansari Road

Daryaganj, New Delhi 110002, India

© 1989 Colin Beardon/Ellis Horwood Limited

**British Library Cataloguing in Publication Data**

Artificial intelligence terminology: a reference guide. —

(Ellis Horwood series in artificial intelligence

1. Artificial intelligence

I. Beardon, Colin

006.3

**Library of Congress CIP data available.**

ISBN 0-7458-0718-6 (Ellis Horwood Limited—Library Edn.)

ISBN 0-7458-0763-1 (Ellis Horwood Limited—Student Edn.)

ISBN 0-470-21601-8 (Halsted Press)

Printed in Great Britain by Hartnolls, Bodmin

**COPYRIGHT NOTICE**

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the permission of Ellis Horwood Limited, Market Cross House, Cooper Street, Chichester, West Sussex, England.

DEF079218



**AI**

[ \* ] AI is an abbreviation of '*Artificial Intelligence*'.

**AKO-link**

[knowledge representation] Generally, an AKO-link ('A-Kind-Of') is another name for an *inst-link* within a *semantic network*. In *KRL*, however, 'AKO-links' cover both *ISA-links* and *inst-links*.

See also: inheritance hierarchy, *inst-link*, *ISA-link*, *semantic network*.

**algorithm**

[software engineering] An algorithm is a specification, written in some notation, for a series of steps which, if carried out, will result in a particular task being performed. It expresses a general method for achieving a particular result, some examples of everyday algorithms being knitting patterns and recipes. Algorithms are used in computing to break down a given problem into a number of steps which can then be written as programming language statements.

See also: program, stepwise refinement, structured programming, top-down design.

**allophone**

[speech processing] Each *phoneme* can be realised in actual speech in a variety of ways depending, for example, on the sounds that precede or follow it. The set of alternative realisations of a phoneme is called an 'allophone'.

See also: phoneme, segmentation.

**alpha-beta pruning**

[game playing] Alpha-beta pruning is a technique applied to improve the efficiency of systems that play two-person *zero-sum* games. It is applicable where a *depth-first search* strategy with limited depth is employed using *evaluation functions*. It is based on the principle that, having fully evaluated one move, other moves are not worth exploring further once a counter-move has been discovered that makes them less attractive than some fully evaluated move. The name 'alpha-beta' acknowledges the fact that a move that is attractive to one player (alpha) is proportionately unattractive to their opponent (beta). The technique requires no extra knowledge to be applied yet can significantly improve efficiency, especially if the most likely move is explored first.

See also: game playing, game tree, minimax.

**antecedent**

[linguistics] The antecedent of an anaphoric expression is the earlier linguistic expression which determines its *reference*.

See also: anaphor, anaphora.

[logic] In a conditional proposition, standardly expressed in the form

*if P then Q*

or in *propositional calculus*

$P \rightarrow Q$

the antecedent is the component proposition (*P*) immediately following the 'if', or, in logical notation, preceding the ' $\rightarrow$ '.

See also: consequent.

[problem solving] The term 'antecedent' can be used to refer to a small procedure that employs *forward chaining* techniques within a predominantly *backward chaining* system.

See also: antecedent rule, antecedent theorem.

**antecedent rule**

[expert systems] An antecedent rule is a rule that performs *forward chaining* within a system that generally employs *backward chaining*. Antecedent rules are usually included for efficiency, to allow certain facts to be explicitly asserted, rather than being repeatedly inferred by the application of rules. Antecedent rules are employed in *MYCIN*.

See also: assert, backward chaining, forward chaining, inference.

**antecedent theorem**

[PLANNER] An antecedent theorem in *PLANNER* behaves like an *if-added demon*. It allows the programmer to specify actions that are to be performed whenever a fact of a particular kind is *asserted*. It can be used to explicitly record certain properties of objects by asserting appropriate facts, thus avoiding excessive use of inferencing where a property is often referred to. For example, an antecedent theorem might assert

animate(X)

whenever a fact of the form

male(X)

was asserted.

See also: assert, if-added demon, inheritance hierarchy, PLANNER.

**connectivity**

[connection science] Connectivity refers to the pattern of *weighted* links between units in a connectionist network. There may be partial connectivity, where some of the units are connected to one another, or there may be complete connectivity, where every unit is connected to every other unit.

**CONNIVER**

[programming languages & problem solving] CONNIVER is a *problem solving* language that was designed to overcome the automatic *backtracking* in *PLANNER* by enabling the programmer to direct the flow of control in the *program*.

See also: backtracking, PLANNER.

Ref: Sussman, G.J. & McDermott, D. (1972) *Why Conniving is better than Planning*. A.I.Memo 255A, MIT, Camb, Mass

**connotation**

[philosophy] J.S.Mill distinguished between the connotation and the *denotation* of a word. In using a word, such as chair, one implies that the things to which it applies have certain properties, such as 'being a piece of furniture' and 'designed for sitting on'. Those properties make up the connotation of the word.

See also: denotation, intension, sense.

**CONS**

[LISP] CONS is a *LISP function* that takes an *S-expression* and a *list* and creates a new list consisting of the S-expression as its *head*, followed by the old list as its *tail*, e.g.

$$\text{cons}('a \ '(b \ c))$$

will produce the list

$$(a \ b \ c)$$

See also: CAR, CDR, dotted pair, LISP, list, quote.

**consequent**

[logic] In a conditional proposition, standardly expressed in the form

$$\text{if } P \text{ then } Q$$

or in *propositional calculus*

$$P \rightarrow Q$$

the consequent is the component proposition (*Q*) immediately following 'then', or, in logical notation, following ' $\rightarrow$ '.

See also: antecedent.

### expectation based parser

[computational linguistics] At various stages when processing text, a top-down *parser* will be faced with a number of tests that can be applied. An expectation-based parser uses what it has already parsed to restrict its choice at this point by only carrying out tests that it has predicted. The entry for each word in the *lexicon* will contain the categories to which it belongs and some expectations about what may appear next and actions to be performed if the expectations are met. The parser is initialised to a *set* of expectations at the start of the sentence before processing commences.

For example, the word 'the' might expect a noun to follow and, if it does, it can record that a noun-phrase has been found. The process of setting up expectations based on the lexical entries for words, seeing if these expectations are satisfied, and carrying out their actions, is repeated until the end of the sentence is predicted and found, or the input does not match any expectation.

See also: categorial grammar, English Language Interpreter, word-expert parser.

### EXPEL

[conceptual dependency] EXPEL is a *primitive act* within *conceptual dependency* representation that represents the expulsion of an object from the body of an animal.

See also: conceptual dependency, primitive act.

### EXPERT

[knowledge representation] EXPERT is a simple *knowledge representation* language designed for general use in representing expert knowledge. It is best suited to diagnosis or classification problems, which is one of the reasons it is most widely used in medical applications. EXPERT is implemented in FORTRAN which makes it portable and efficient.

See also: KRL.

Ref: Weiss, S. M., Kulikowski, C. A. (1979) EXPERT: a system for developing consultation models. In: *Proceedings of the sixth IJCAI*, Tokyo

### expert system

[ \* ] An expert system is a computer *program* which attempts to embody the knowledge and decision-making facilities of a human expert in order to carry out a task generally regarded as requiring some degree of human expertise. Expert systems usually operate within restricted domains, some well known examples being medical diagnosis and mineral exploration. Their reasoning is essentially *heuristic*, in that an *algorithmic* approach which is guaranteed to produce an *optimal* solution to a problem, is not feasible. An expert system will generally



consist of a *rule base*, an *inference engine* and a user interface (which will generally provide an explanation facility).

See also: CADUCEUS, CENTAUR, DENDRAL, INTERNIST, knowledge-based system, MACSYMA, MYCIN, PROSPECTOR, rule base, shell, TAXMAN, XCON.

## explanation-based learning

[machine learning] Explanation-based learning (EBL) involves the learning of a concept from a single example. The *domain* theory held by the system is used to explain why the example is an instance of the concept under study. In explanation-based generalisation, a subset of EBL, this explanation is generalised so that the resulting rule may be applied to similar, though not necessarily identical, situations.

See also: concept learning, learning by induction, similarity-based learning.

## exponential

[mathematics] The term 'exponential' in mathematics indicates that a term is raised to a power (normally greater than 1). For example, '2<sup>3</sup>' is expressed exponentially, whereas '8' is not.

[software engineering] The term exponential is used to describe algorithmic *complexity*. An *algorithm* whose complexity for *n* inputs is measured in terms of  $C^n$ , for some constant *C*, is called an exponential algorithm. Most exponential algorithms are not feasible for any but the smallest input data problems.

See also: algorithm, combinatorial explosion, complexity, order of complexity, polynomial.

## extended Horn clause

[logic] For the purposes of practical *logic programming*, *Horn clauses* are over-restrictive and some extensions to Horn clause notation have been adopted by logic programming languages such as *PROLOG*. For example, a Horn clause, by definition, allows at most one unnegated *literal*. When a Horn clause is converted to *PROLOG* notation this means that there can be no negated terms in the *body* of the *PROLOG clause*. To get around this limitation a special form of *negation* (called *negation-as-failure*) is introduced that can be applied to terms in the body of a clause. This is one way in which *PROLOG* notation is an extension of Horn clause notation, others include the use of operators such as *cut*.

See also: clause, cut, Horn clause, negation-as-failure.

## inductive logic

[logic] In an inductive logic the conclusion follows from the premises only with a degree of probability and an additional premise may change the probability of the conclusion. The *conjunction* of the premises and the *negation* of the conclusion is not a contradiction.

Thus typically one may infer from a finite number of statements such as "*raven*<sub>1</sub> is black", "*raven*<sub>2</sub> is black" ... "*raven*<sub>*n*</sub> is black", the conclusion "all ravens are black". Clearly the conclusion only follows with a certain probability and the addition of the new premise "*raven*<sub>*n*+1</sub> is not black" would invalidate the *inference*. Inductive logic is contrasted with *deductive logic*.

See also: BACON, concept learning, deductive logic, learning by induction, non-monotonic reasoning.

## inference

[logic] An inference is the drawing of a conclusion, or alternatively is just the conclusion drawn. A logical inference can be expressed as a relationship among a set of sentences or propositions.

See also: argument, deductive logic, inductive logic.

## inference engine

[expert systems] An inference engine is that part of an expert system that interprets rules and facts in order to make *inferences*. Depending upon the system, it may be driven by a specific question (*backward chaining*) or attempt to make inferences from a set of *data* (*forward chaining*). Often a combination of both methods is employed.

See also: chaining, expert system, inference, rule base.

## infinite loop

[software engineering] An infinite loop is a series of programming language statements which will repeat indefinitely, usually due to an error in the *program* code. The failure to alter a control *variable* within an iterative statement is a typical cause of infinite loops. For example, if  $x = 1$ , then the loop,

```
while x <= 10 do write ('hello')
```

will repeat indefinitely.

See also: tractable.

## kludge

[ \* ] A kludge is a part of a mechanism whose behaviour cannot be simply explained by reference to the principles adopted by the rest of the mechanism, or by universal principles. Kludges can be best explained by a number of *domain-specific* rules. Insofar as the mechanism operates within a *paradigm*, the term 'kludge' can refer to aspects of its behaviour that are better explained by theories external to the paradigm. For example, aspects of human behaviour may be better explained in terms of human evolution, rather than as integral aspects of a *symbolic information processing* system.

## knowledge acquisition

[expert systems] Knowledge acquisition is the activity of attempting to make explicit the relevant knowledge of a human expert. Often the objective is to encode the expertise of the human expert in terms of the formal notation of an expert system (e.g. rules, *metarules* and facts). This can be broken down into three stages: eliciting knowledge, encoding it in some *canonical representation*, and validating it by making sample *inferences* which can be checked by domain experts and give rise to amendments.

See also: expert system, knowledge elicitation, knowledge engineer, TEIRESIAS, validation.

## knowledge based system

[ \* ] A knowledge based system (KBS) can be contrasted to a *database* or *information retrieval* system. In the latter cases information is retrieved from a store in response to precise requests and only the information described is presented. In KBS a more general goal can be stated and the system may make any relevant *inferences* in order to provide an appropriate answer. The result may not carry absolute certainty, but should represent an intelligent response to a problem given the knowledge available to the system.

The term 'Knowledge based systems' is sometimes used synonymously with '*expert systems*', though it is usually taken to be a more general term incorporating, for instance, some *natural language processing* systems.

See also: database system, expert system, information retrieval.

## knowledge elicitation

[expert systems] Knowledge elicitation is the task of trying to make explicit the knowledge behind the performance of experts in their particular *domain*. One method is to ask the experts to explicitly state the rules they work by, but this does not always work well as their expertise is essentially performative, rather than *declarative*. An alternative approach is for the expert to give examples of their reasoning and for rules to be derived from these. In other cases, rules may be found ready formalised in text books or similar literature.

See also: expert system, knowledge acquisition, knowledge engineer.

### production system

[expert system] A production system is an *inference* system that consists of three parts: the working memory, which represents the *current state* of the inference (also known as the 'context' or 'short-term memory buffer'); the production memory, which represents the inference rules; and the rule *interpreter*, which applies the rules to the working memory. At each cycle all of the rules in the production memory are examined to see if their conditions are satisfied in the working memory and a list of rules that can be fired is produced. A *conflict resolution* strategy is then employed to decide which rule to fire. The action part of the chosen rule is executed and the working memory is updated as a result. This process continues until either the working memory contains the *goal*, or no more rules can be fired.

See also: DENDRAL, MYCIN, OPS5, production rule, PROSPECTOR.

### program

[computer systems] A computer program consists of a series of definitions and/or instructions conforming to the *syntax* of a given programming language which, when executed on, or interpreted by, a computer will perform a certain task. A program is often defined as being a marriage between *data structures* and *algorithms*. Because programs will be executed on some machine, they are essentially *procedural*, yet insofar as they describe the problem in terms of its own *domain*, they may also aspire to being *declarative* or functional.

See also: algorithm, data structure, declarative, procedural.

### program synthesis

[problem solving & programming] Program synthesis (also known as *automatic programming*) involves the generation of computer program code given some description of the problem in another form. This description might be in the form of examples of behaviour, a statement in a *formal language* (such as *logic*), a description in natural language, or result from an interactive dialogue.

## PROGRAMMAR

[computational linguistics & programming languages] PROGRAMMAR is a *natural language parsing* system developed by T. Winograd for the SHRDLU project. Though formally similar to an ATN, PROGRAMMAR is based upon ideas from *systemic grammar*. The language to be parsed is defined *procedurally*, enabling special purpose mechanisms to be introduced if required. It attempts to be as deterministic as possible, though facilities do exist for *backtracking* and trying alternative paths.

See also: augmented transition network, deterministic parsing, SHRDLU, systemic grammar.



## ROSIE

[expert systems] ROSIE is an artificial, English-like language for expressing facts and *inference* rules within expert systems. Though some of its constructs are quite formal, it can be read and interpreted *declaratively* once certain conventions are learned.

## rote learning

[learning] Rote learning refers to knowledge accumulation involving no transformations or processing of the information received. Forms of rote learning include learning by being programmed and learning by memorisation.

See also: learning from instruction.

## rule base

[expert systems] Just as a *database* is essentially a collection of data, so a rule-base is a collection of rules. Typically the term is used to refer to the set of rules in an *expert system*.

See also: production memory.

## rule-based system

[expert systems] A rule-based system is any system that represents knowledge as a set of *rules* that can be interpreted in a uniform way by applying them to known *facts* to infer new facts. The objective of a rule-based system is to encode practical human knowledge in such a way that it can mimic the *problem-solving* abilities of a human expert.

See also: expert system, inference.

## runtime error

[programming] A runtime error is one which occurs after *compilation*, during the execution of the translated source program. This sort of error is often due to a *semantic* error or a memory violation.

See also: bug, compilation error, debugging, semantic.

## runtime stack

[programming languages] A runtime stack is a mechanism for giving a *procedural* interpretation of the way that one process can call another during the execution of a *program*. Any language that allows the recursive calling of processes (e.g. *procedures* in Pascal, *clauses* in *PROLOG*, *functions* in *LISP*) will need to keep track of suspended processes and their return points, and a *stack* is an obvious choice for this.

See also: clause, function, nondeterminism, procedure, recursion, stack.

**word recognition**

[cognitive psychology] The study of word recognition is the study of the mental processes involved in identifying the meanings of words. Word recognition involves encoding the initial percept of the word, locating the appropriate entry in the *mental lexicon*, and selecting the appropriate meaning for a given context. Several alternative models of word recognition have been proposed.

See also: lexical access, logogen model, mental lexicon.

**working memory**

[cognitive psychology] Working memory refers to that part of human memory that is sometimes called *short term memory*. Working memory is a transient store of information that can be maintained only through rehearsal. Use of the term 'working memory' emphasises the fact that the information is being held for use by mental procedures. Information can be retrieved from *long term memory* and held active in working memory as various computations are completed.

See also: long term memory, short term memory.

**workstation**

[computer systems] Workstations are usually stand-alone, single-user computers, but their power is such that they can normally support many users. Workstations sometimes lend themselves to specific applications, such as graphics and large scale numerical computation.

**WPE**

[programming languages] WPE is an abbreviation of 'Warren Prolog Engine' which is the same thing as the *Warren Abstract Machine*.

1 Teresa M. Corbin (SBN 132360)  
Thomas Mavrakakis (SBN 177927)  
2 HOWREY SIMON ARNOLD & WHITE, LLP  
301 Ravenswood Avenue  
3 Menlo Park, California 94025  
Telephone: (650) 463-8100  
4 Facsimile: (650) 463-8400

5 Attorneys for Plaintiff SYNOPSIS, INC.  
and for Defendants AEROFLEX INCORPORATED,  
6 AMI SEMICONDUCTOR, INC., MATROX  
ELECTRONIC SYSTEMS, LTD., MATROX  
7 GRAPHICS INC., MATROX INTERNATIONAL  
CORP. and MATROX TECH, INC.  
8

9 UNITED STATES DISTRICT COURT  
10 NORTHERN DISTRICT OF CALIFORNIA  
11 SAN FRANCISCO DIVISION  
12

13 RICOH COMPANY, LTD., )

14 Plaintiff, )

15 vs. )

16 AEROFLEX INCORPORATED, et al., )

17 Defendants. )

Case No. C03-04669 MJJ (EMC)

Case No. C03-2289 MJJ (EMC)

**NOTICE OF MANUAL FILING OF  
EXHIBIT 15 TO RESPONSIVE CLAIM  
CONSTRUCTION BRIEF**

18 SYNOPSIS, INC., )

19 Plaintiff, )

20 vs. )

21 RICOH COMPANY, LTD., a Japanese )  
corporation )

22 Defendant. )  
23

Date: October 29, 2004

Time: 9:30 AM

Courtroom: 11

Judge: Martin J. Jenkins

**MANUAL FILING NOTIFICATION**

Regarding: Exhibit 15 to Responsive Claim Construction Brief

This filing is in paper or physical form only, and is being maintained in the case file in the Clerk's office. If you are a participant in this case, this filing will be served in hard-copy shortly. For information on retrieving this filing directly from the court, please see the court's main web site at <http://www.cand.uscourts.gov> under Frequently Asked Questions (FAQ).

This filing was not efiled for the following reason(s):

☒ Voluminous Document (PDF file size larger than the efile system allows)

☐ Unable to Scan Documents

☐ Physical Object (description): \_\_\_\_\_

☐ Non-Graphic/Text Computer File (audio, video, etc.) on CD or other media

☐ Item Under Seal

☐ Conformance with the Judicial Conference Privacy Policy (General Order 53).

☐ Other (description): \_\_\_\_\_



MICROSOFT PRESS®

# COMPUTER DICTIONARY



THE COMPREHENSIVE  
STANDARD FOR  
BUSINESS, SCHOOL,  
LIBRARY, AND HOME

**Microsoft**  
P R E S S

DEF083323

PUBLISHED BY  
Microsoft Press  
A Division of Microsoft Corporation  
One Microsoft Way  
Redmond, Washington 98052-6399

Copyright © 1991 by Microsoft Press, a division of Microsoft Corporation.

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Library of Congress Cataloging-in-Publication Data  
Microsoft Press computer dictionary : the comprehensive standard for business, school, library, and home.

p. cm.

ISBN 1-55615-231-0

1. Computers--Dictionaries. 2. Microcomputers--Dictionaries.

I. Microsoft Press.

QA76.15.M54 1991

004.16'03--dc20

91-9904

CIP

Printed and bound in the United States of America.

2 3 4 5 6 7 8 9 MLML 6 5 4 3 2 1

Distributed to the book trade in Canada by Macmillan of Canada, a division of Canada Publishing Corporation.

Distributed to the book trade outside the United States and Canada by Penguin Books Ltd.

Penguin Books Ltd., Harmondsworth, Middlesex, England  
Penguin Books Australia Ltd., Ringwood, Victoria, Australia  
Penguin Books N.Z. Ltd., 182-190 Wairau Road, Auckland 10, New Zealand  
British Cataloging-in-Publication Data available.

**Acquisitions Editor:** Marjorie Schlaikjer

**Project Editor:** Mary Ann Jones

**Technical Editors:** David Rygmyr, Jeff Hinsch, Mary DeJong, Dail Magee, Jr.

**Manuscript Editor:** Pamela Beason

**Copy Editor:** Alice Copp Smith

DEF083324



lated on 8086/8088 or 80286 machines as well, but the performance is poor. *Compare* conventional memory, extended memory.

**expansion** A means of increasing the capabilities of a microcomputer by adding hardware designed to perform a task that is not built into the basic system. *Expansion* is generally used in reference to the addition of printed circuit boards (expansion boards) that plug into openings (expansion slots) inside the body of the computer. In IBM personal computers and others designed as open-ended systems, these slots enable expansion boards and associated devices to connect with and transfer information on the computer's main data highway, the bus. Computers with expansion slots can be equipped with as many additional pieces of hardware as there are slots available. *See also* expansion slot, open architecture.

**expansion board** A circuit board holding chips and other electronic components connected by conductive paths that is plugged into a computer's bus (main data-transfer path) to add functions or resources to the computer. Typical expansion boards add memory, disk-drive controllers, video support, parallel and serial ports, and internal modems. The simple terms *board* and *card* are used interchangeably by most people to refer to all expansion boards. *See also* expansion slot.

**expansion bus** *See* AT bus.

**expansion slot** A socket inside a computer console, designed to hold expansion boards and connect them to the system bus (data pathway). Most personal computers have from three to eight expansion slots, with the notable exceptions of the Apple Macintosh and Macintosh Plus, which have none, and the Macintosh SE, which has one. Expansion slots provide a means of adding new or enhanced features or more memory to the system. *See also* expansion board.

**expert system** A type of application program that makes decisions or solves problems in a particular field, such as finance or medicine, by using knowledge and analytical rules defined by experts in the field. Human experts solve problems by using a combination of factual knowledge and reasoning ability. In an expert system, these two

essentials are contained in two separate but related components, a knowledge base and an inference engine. The knowledge base provides specific facts and rules about the subject, and the inference engine provides the reasoning ability that enables the expert system to form conclusions. Expert systems also provide additional tools in the form of user interfaces and explanation facilities. User interfaces, as with any application, enable people to form queries, provide information, and otherwise interact with the system. Explanation facilities, an intriguing part of expert systems, enable the systems to explain or justify their conclusions, and they also enable developers to check on the operation of the systems themselves. Expert systems originated in the 1960s; fields in which they are used include chemistry, geology, medicine, banking and investments, and insurance. *See also* artificial intelligence, intelligent database.

**exploded view** A type of display that shows a structure with its parts separated but drawn in relation to one another. *See* the illustration. A charting program, for example, could create an exploded pie chart by separating one segment from the rest of the pie. Similarly, an engineering or architectural drafting program could display a structure with its parts exploded outward so that the designer can view the pieces separately but as parts of the whole.

**exponent** In mathematics, a number that shows how many times another number is used as a factor in a calculation. Although exponents are commonly thought of as representing higher powers of a number, only positive exponents, as in  $2^3$ , indicate multiplication (2 times 2 times 2). Negative exponents, as in  $2^{-3}$ , indicate division (1 divided by  $2^3$ ), and fractional exponents, as in  $8^{1/3}$ , indicate the root of the number (here the cube root of 8).

In the floating-point, or exponential, notation commonly used with computers, the exponent is a number that indicates the power of 10 to be multiplied by the number in the fixed-point portion of the notation. Essentially, the exponent in a floating-point number shows the number of places the decimal point is moved to the right (positive exponent) or to the left (negative exponent) when the number